TRABAJO DE FIN DE MÁSTER EN

INGENIERÍA INFORMÁTICA

CURSO 2014-2015

ARQUITECTURA PARA LA MEJORA DE LA CALIDAD DE SERVICIOS MULTIMEDIA EN REDES DEFINIDAS POR SOFTWARE

Marco Antonio Sotelo Monge

Directores:

Luis Javier García Villalba

Ana Lucila Sandoval Orozco

Departamento de Ingeniería del Software e Inteligencia Artificial

Convocatoria de Septiembre

Calificación: 10 – Matrícula de Honor

MÁSTER EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

El abajo firmante, matriculado en el Máster en Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "***Arquitectura para la Mejora de la Calidad de Servicios Multimedia en Redes Definidas por Software***", realizado durante el curso académico 2014-2015 bajo la dirección de Luis Javier García Villalba y de Ana Lucila Sandoval Orozco en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Marco Antonio Sotelo Monge

Luis Javier García Villalba

Ana Lucila Sandoval Orozco

Abstract

The quality of multimedia transmissions in software defined networking depends on the capability to monitor the actual state of the network. Quality of service evaluation usually represents a critical task, performed by multimedia transmission protocols at run time. However, the information provided by those protocols does not take part of the deployed actions to improve network conditions at lower levels of the infrastructure. This research proposes a new SDN architecture to improve the user quality of experience (QoE). The above-mentioned architecture is composed by software elements that take advantage of the information provided by high level multimedia protocols in order to improve routing decisions through the use of algorithms executed in the SDN controller. The architecture also proposes the creation of a communication protocol that facilitates the information exchange between a host and the SDN controller. Furthermore, it proposes a set of software components in the application and control layers. Emulation tests performed in the experimentation phase show that user quality of experience is improved, regarding to the perceived multimedia service.

Keywords

API, Controller, Jitter, MOS, Multimedia Transmission, OpenFlow, PSNR, QoE, Quality of Experience, Routing, Software Defined Networking.

Resumen

La calidad de las transmisiones multimedia en redes definidas por software depende de la capacidad de monitorizar el estado actual de la red. La evaluación de la calidad de servicio representa una tarea importante y se realiza en tiempo de ejecución por los protocolos de transmisión multimedia. No obstante, la información proporcionada por tales protocolos no se tiene en cuenta en las acciones de mejora de las condiciones de red en capas inferiores de la infraestructura. Este trabajo propone una nueva arquitectura SDN cuyo objetivo es mejorar la calidad de experiencia de usuario (QoE). Dicha arquitectura está compuesta de elementos de software capaces de aprovechar la información proporcionada por los protocolos multimedia de alto nivel para mejorar las decisiones de encaminamiento mediante algoritmos ejecutados en un controlador SDN. La arquitectura propuesta plantea también la creación de un protocolo de comunicación que facilita el intercambio de información entre un host y un controlador SDN. Además, propone un conjunto de componentes de software en las capas de control y aplicación. Las pruebas de emulación realizadas en la fase de experimentación demuestran una mejora en la calidad de experiencia del usuario con respecto al servicio multimedia recibido.

Palabras clave

API, Calidad de Experiencia, Controlador, Encaminamiento, Jitter, MOS, OpenFlow, PSNR, QoE, Red Definida por Software, Transmisión Multimedia.

Agradecimientos

Expreso mi agradecimiento a las personas que me han ayudado en la realización del presente trabajo. En particular quiero agradecer a mis directores, Luis Javier García Villalba y Ana Lucila Sandoval Orozco, por la dedicación esmerada para la consecución del trabajo, así como por el tiempo y recursos de investigación que hicieron posible el logro de los objetivos planteados.

De manera especial, quiero expresar también mi agradecimiento a Leonardo Valdivieso y Lorena Barona, quienes colaboraron conmigo en las tareas de investigación, orientándome y apoyándome sin escatimar esfuerzo alguno en todas las ocasiones que recurrí a ellos.

Asimismo, doy las gracias a mi madre, quien me ha educado dándome siempre mucho más de lo que la vida le ha permitido; y a mis hermanos, por su apoyo incondicional a lo largo de mi existencia.

Finalmente, agradezco al Programa Nacional de Becas y Crédito Educativo del Ministerio de Educación de Perú, por haber financiado mis estudios de Máster mediante la Beca de Excelencia para Estudios de Postgrado de la Convocatoria 2013-III.

ÍNDICE GENERAL

ÍNDICE GENERAL IX
ÍNDICE DE FIGURAS XI
ÍNDICE DE TABLAS XIII
ÍNDICE DE ABREVIATURAS XV

1. INTRODUCCIÓN 1

1.1. OBJETO DE LA INVESTIGACIÓN 2
1.2. TRABAJOS RELACIONADOS 3
1.3. ESTRUCTURA DEL TRABAJO 5

2. REDES DEFINIDAS POR SOFTWARE 7

2.1. DEFINICIÓN 7
2.2. HISTORIA DE LAS REDES DEFINIDAS POR SOFTWARE 9
2.3. SEPARACIÓN DE PLANOS 11
2.3.1. EL PLANO DE CONTROL 11
2.3.2. EL PLANO DE DATOS 12
2.4. VENTAJAS DE LA SEPARACIÓN DE PLANOS 13
2.5. EJEMPLOS DE LA SEPARACIÓN DE PLANOS 14
2.5.1. RCP (ROUTING CONTROL PLATFORM) 14
2.5.2. ETHANE 15
2.5.3. OPENFLOW 16
2.6. ÁREAS DE APLICACIÓN DE SDN 16
2.7. DESAFÍOS DE LA TECNOLOGÍA SDN 18

3. EL PROTOCOLO OPENFLOW 21

3.1. INTRODUCCIÓN 21
3.2. CONMUTADOR OPENFLOW 22
3.3. TABLAS DE FLUJO 23
3.3.1. CAMPOS DE ENCABEZADO 23
3.3.2. CONTADORES 24
3.3.3. ACCIONES 25
3.3.3.1. ACCIONES OBLIGATORIAS 26
3.3.3.2. ACCIONES OPCIONALES 26
3.4. FLUJO DE CORRESPONDENCIA (*MATCHING*) 27
3.5. CANAL SEGURO 28
3.5.1. TIPOS DE MENSAJES OPENFLOW 28
3.5.1.1. CONTROLADOR A CONMUTADOR 28
3.5.1.2. ASÍNCRONOS 29
3.5.1.3. SIMÉTRICOS 30
3.5.2. CARACTERÍSTICAS DE LA CONEXIÓN SEGURA 30

4. CONTROLADORES SDN 33

4.1. INTRODUCCIÓN 33
4.2. MODELO DE CONTROLADOR IDEALIZADO 33
4.3. CONTROLADORES DE RED EN LA INDUSTRIA 35
4.3.1. FLOODLIGHT 35
4.3.1.1. MÓDULOS DEL CONTROLADOR 36
4.3.1.2. MÓDULOS DE APLICACIONES 37
4.3.2. OPENDAYLIGHT 38
4.3.3. NOX/POX 40
4.3.4. TREMA 41

4.3.5. RYU 42
4.3.6. MININET 43
4.4. COMPARATIVA DE CONTROLADORES 45

5. COMUNICACIÓN MULTIMEDIA EN TIEMPO REAL 47

5.1. INTRODUCCIÓN 47
5.2. PROTOCOLOS MULTIMEDIA 47
5.2.1. RTSP 47
5.2.2. RTP 50
5.2.2.1. ENCABEZADO RTP 50
5.2.2.2. FUNCIONAMIENTO DEL PROTOCOLO RTP 52
5.2.3. RTCP 52
5.2.3.1. ENCABEZADO RTCP 53
5.2.3.2. FUNCIONES DEL PROTOCOLO RTCP 54
5.3. MEDICIÓN DE LA CALIDAD DE UN VÍDEO 55
5.3.1. PSNR 55
5.3.2. MOS 56

6. ARQUITECTURA PARA LA MEJORA DE LA CALIDAD MULTIMEDIA 59

6.1. INTRODUCCIÓN 59
6.2. DESCRIPCIÓN DE LA ARQUITECTURA 59
6.2.1. CAPA DE INFRAESTRUCTURA 60
6.2.2. CAPA DE CONTROL 60
6.2.3. CAPA DE APLICACIÓN 61
6.3. EL PROTOCOLO PCMS 61
6.3.1. ANTES DE LA TRANSMISIÓN 62
6.3.2. DURANTE DE LA TRANSMISIÓN 63
6.3.3. DESPUÉS DE LA TRANSMISIÓN 64
6.4. COMPONENTES DE LA ARQUITECTURA Y SU IMPLEMENTACIÓN 64
6.4.1. COMPONENTES MULTIMEDIA 64
6.4.1.1. SOFTWARE MULTIMEDIA 65
6.4.1.2. AGENTE DE GESTIÓN DE RECURSOS (AGR) 66
6.4.1.3. SERVICIO DE GESTIÓN DE RECURSOS (SGR) 67
6.4.2. COMPONENTES DEL CONTROLADOR SDN 67
6.4.2.1. MONITOR DE RED (MR) 68
6.4.2.2. BUSCADOR DE RUTAS (BR) 69

7. EXPERIMENTOS Y RESULTADOS 73

7.1. COMPONENTES SOFTWARE 73
7.2. ENTORNO DE EJECUCIÓN 73
7.3. EXPERIMENTOS 74
7.3.1. TOPOLOGÍA DE PRUEBA PARA EL PROTOCOLO PCMS 74
7.3.2. TOPOLOGÍA DE PRUEBA PARA LA TRANSMISIÓN DE VÍDEO 78
7.4. RESULTADOS 79

8. CONCLUSIONES Y TRABAJO FUTURO 83

8.1. CONCLUSIONES 83
8.2. TRABAJO FUTURO 85

BIBLIOGRAFÍA 91

ÍNDICE DE FIGURAS

Figura 2.1: Arquitectura SDN. 8
Figura 2.2: Arquitectura RCP. 14
Figura 2.3: Ejemplo de comunicación en una red Ethane. 15
Figura 3.1: Componentes de un conmutador OpenFlow. 22
Figura 3.2: Flujo de un paquete en un conmutador OpenFlow v1.0.0. 27
Figura 4.1: Modelo de controlador idealizado. 34
Figura 4.2: Arquitectura de Floodlight y sus aplicaciones. 36
Figura 4.3: Arquitectura de OpenDaylight. 39
Figura 4.4: Relaciones entre módulos de usuario y de sistema en Trema. 42
Figura 4.5: Red OpenFlow creada en Mininet. 44
Figura 5.1: Comunicación RTSP entre un cliente y un servidor. 49
Figura 5.2: Encabezado de un paquete RTP. 50
Figura 5.3: Encabezado de un paquete RTCP tipo RR. 54
Figura 6.1: Arquitectura para la mejora de la calidad multimedia en redes SDN. 60
Figura 6.2: Descripción del protocolo PCMS. 62
Figura 6.3: Componentes multimedia de la arquitectura. 65
Figura 6.4: Esquema de operación en Video Tester. 66
Figura 6.5: Componentes del controlador SDN. 68
Figura 7.1: Topología creada con la clase TreeNet. 75
Figura 7.2: Topología creada en el caso de prueba nº 8. 77
Figura 7.3: Topología de prueba para la transmisión de vídeo. 78
Figura 7.4: Medida del PSNR sin la arquitectura propuesta. 80
Figura 7.5: Medida del PSNR después de implementar la arquitectura. 80
Figura 7.6: Medida del MOS sin la arquitectura propuesta. 81
Figura 7.7: Medida del MOS luego de implementar la arquitectura. 82

ÍNDICE DE TABLAS

Tabla 3.1: Estructura de una entrada en una tabla de flujo. 23
Tabla 3.2: Campos de un paquete utilizados para crear criterios de correspondencia. 23
Tabla 3.3: Longitudes de campo y la forma en que son aplicados en las entradas de flujo. 24
Tabla 3.4: Lista de contadores requeridos para su uso en mensajes de estadísticas. 25
Tabla 4.1: Comparación de controladores SDN. 45
Tabla 5.1: Escala de valores del MOS [KRW03]. 56
Tabla 5.2: Equivalencia entre el MOS y PSNR [KRW03]. 57
Tabla 6.1: Paquetes de vídeo en el Buscador de Rutas. 71
Tabla 7.1: Componentes software para la evaluación del modelo. 73
Tabla 7.2: Parámetros de configuración de topologías en Mininet. 75
Tabla 7.3: Topologías y tiempos de procesamiento en el RMS y AGR. 76
Tabla 7.4: Parámetros de cada enlace en la topología de prueba. 78
Tabla 7.5: Característica del vídeo empleado en las pruebas. 79

ÍNDICE DE ABREVIATURAS

ACL	*Access Control List*
AGR	*Agente de Gestión de Recursos*
API	*Application Program Interface*
AS	*Autonomous System*
ASIC	*Application-Specific Integrated Circuit*
BGP	*Border Gateway Protocol*
CPU	*Central Processing Unit*
DARPA	*Defense Advanced Research Projects Agency*
FPGA	*Field-Programmable Gate Array*
GPU	*Graphics Processor Unit*
GUI	*Graphical User Interface*
HTTP	*Hypertext Transfer Protocol*
HVS	*Human Visual System*
ICMP	*Internet Control Message Protocol*
IEEE	*Institute of Electrical and Electronics Engineers*
IGP	*Interior Gateway Protocol*
IP	*Internet Protocol*
IPC	*Inter-Process Communication*
ISP	*Internet Service Provider*
IXP	*Internet Exchange Point*
MFFE	*Mobile Flow Forwarding Engine*
MOS	*Mean Opinion Score*

MPEG	*Moving Picture Experts Group*
NCP	*Network Control Point*
NFV	*Network Functions Virtualization*
NOS	*Network Operating System*
ONF	*Open Networking Foundation*
OSI	*Open System Interconnection*
PCEP	*Path Computation Element Communication Protocol*
PCMS	*Protocolo de Control Multimedia para SDN*
PSNR	*Peak Signal-to-Noise Ratio*
QoE	*Quality of Experience*
QoS	*Quality of Service*
RAM	*Random Access Memory*
RCP	*Routing Control Platform*
RCS	*Route Control Server*
REST	*Representational State Transfer*
RFC	*Request for Comments*
RTCP	*RTP Control Protocol*
RTP	*Real Time Protocol*
RTSP	*Real Time Streaming Protocol*
SAL	*Service Adaption Layer*
SDMN	*Software Defined Mobile Network*
SDN	*Software Defined Networking*
SDX	*SDN-Based Exchange*
SGR	*Servicio de Gestión de Recursos*

TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UDP	*User Datagram Protocol*
VLAN	*Virtual Local Area Network*
VoIP	*Voice over IP*
WAN	*Wide Area Network*

INTRODUCTION

Software Defined Networking is an emerging network architecture where network control is programmable and is separated from packet forwarding functions, which belong to lower levels of the OSI model.

This migration of the control plane, previously inherent to individual network devices, makes it possible to abstract the network infrastructure as a logical entity, thus makes it available to services and applications which can manage the network centrally, and as a single virtual entity.

The centralized control of the network and the separation of data and control planes, inherent to software defined networks, have opened a new business model based on the customization of network services. Multimedia services have been made essential in our lives, concerning a wide range of applications developed due to the motivation of interaction capabilities promoted by this kind of technologies. This communication need has generated a considerable increment of Internet multimedia traffic, especially in the last years. It is expected that this growing trend will be kept [Cis15].

Nowadays, it is not enough the delivery of a multimedia service if its delivery quality is unknown. Users' demands are higher as well as the platforms from which they access to multimedia services. Concepts like Quality of Service (QoS) or Quality of Experience (QoE) have been adopted in several areas of technology, including also multimedia technologies. Because of the nature of their services, and particularly when talking about live transmissions, real time quality measurement represents an important challenge to cope with the available technology. Modern communication protocols rely on these facts from the application point of view, without taking into account the information about the network environment which is usually extracted from the lower layers of the OSI model. From this perspective, software defined networks

provide enough elements that allow applications to know about network state and also allow them to control network behavior by the use of programming interfaces that give them the enough flexibility that is not possible to find in traditional network architectures until these days.

Research Objective

The efficient operation of multimedia services over SDN networks is dependent of the capability to monitor state of the network. Typical solutions are focused on the estimation of network level variables in the data plane (data rate, loss, delay) [VADK14]. The controller can use of this information to reconfigure the network and balance data load between different users. However, the controller does not know about users' quality perception regarding to multimedia data transmitted. A deficient network management decreases the user quality of experience, and at the same time increases processing load in terminal devices which try to keep the same quality of services level, being required to execute high level algorithms (such as error correction algorithms). It is also well known that overcome deficiencies in lower levels reduce processing tasks in upper layers.

This research presents the design of an SDN architecture to improve the users' quality of experience regarding to the multimedia services delivered over a software defined network. For that purpose, the SDN controller is provided by updated information originated by high level multimedia protocols which will be used to improve packet routing decisions in the OpenFlow switches. The proposed architecture allows terminal devices (hosts, mobiles, tablets) to exchange information in the SDN application plane. Furthermore, the PCMS (Multimedia Control Protocol for SDN) protocol is proposed in order to provide the controller updated and real time information about the ongoing multimedia transmission quality.

With the implementation of the architecture through software agents, the PCMS protocol and the modules in the network controller, along with the use of video analysis and network traffic tools, and also an emulated network environment, the effectiveness of the architecture will be demonstrated with the performed experiments in the last chapter.

Related work

Recent works have been extended the scope of traffic management tasks to terminal devices. In [SYF+15] the advantages of software defined networks are taken to extend some management tasks, traditionally performed by security servers (as firewalls), to end devices which, for example, collect network performance statistics and transmit them through programming interfaces to other nodes that process the information from a global perspective. It demonstrates that management tasks can be programmable (either centralized or distributed) in an SDN network.

Network monitoring is an important task in order to get parameters that expose network state. Those parameters depend on the information collected from the data plane. In [VADK14] network monitoring over an SDN network is covered, deploying for that purpose a centralized model based on the use of Application Programming Interfaces (APIs) to get network parameters from the data plane, their processing and consolidation in the control plane, and their communication to the application plane. Thus, it allows other software entities to benefit from the information processed at the controller, provided with the appropriate data exchange interfaces (as REST API).

Video quality evaluation over an IP network is covered on [NOALS12]. It introduces a model based on several metrics that are used to evaluate video quality. The analysis is performed after multimedia content transmission (offline analysis), so that, focusing on the result instead of the process. Quality

evaluation reports are obtained by using Video Tester tool, developed with that purpose. Video Tester has components that distinguish content transmission functions (implemented by multimedia protocols) from those related to data analysis and the before mentioned quality evaluation.

User's Quality of Experience (QoE) [LCMP+13] is a new concept that gained importance in the last years. QoE extends the traditional concept of quality of service by considering also the user's acceptance level with respect to the perception of a particular multimedia service. One of the main analysis areas of QoE is the delivery of multimedia content through mechanisms such as streaming, broadcasting, files transfer, etc. In some cases (such as in video transmission) the measurement of quality of experience in real time becomes essential to prove that a multimedia service fulfills user quality expectations.

In [PF14] and [KSKD+12] quality improvement methods through route optimization are explored. They define some elements of control as part of the SDN architecture, focused on the improvement of multimedia quality of experience based on route optimization in a software defined network. The first one, contrasts the results against quality measurement metrics. The second one, defines an architecture based on service negotiation and is proposed as a work in progress that introduces concepts such as architecture, quality and routes assignment in the transmission of multimedia data.

Document structure

This work is divided in seven chapters. In chapter 2, the underlying concepts related to software defined networks and their characteristics are described.

Chapter 3 explores the operation details of the OpenFlow protocol and its relevance in the development of SDN technology. In particular, it describes the interaction between the switches and the network controller, through the exchange of OpenFlow messages and the corresponding configuration of flow

tables in the devices.

Chapter 4 makes a revision of the most important SDN controllers of the industry. It analyzes their most important characteristics, mainly, from an architecture perspective. A comparison between the inspected controllers is made at the end of the chapter.

Chapter 5 describes RTSP, RTP and RTCP protocols, used for real time multimedia transmissions. Currently, these protocols are used extensively and serve as a basis for the deployment of experiments in this work.

Chapter 6 explains the proposed architecture in order to improve multimedia services quality. It explains the software components and their performed role in the control and application planes of the SDN architecture. Furthermore, the PCMS (Multimedia Control Protocol for SDN) protocol is presented, which has been proposed for the exchange of messages between the architecture components.

Chapter 7 presents the experiments deployed to demonstrate the operation of the proposed architecture, as well as the results obtained after the evaluation of user quality of experience by the use of specialized software.

Finally, chapter 8 presents the conclusions of this work, and the possible lines of future work intended to improve the investigation are mentioned.

1. INTRODUCCIÓN

Una Red Definida por Software (en inglés, *Software Defined Networking* o SDN) es una arquitectura de red emergente donde el control de la red es programable y está separada de las funciones de reenvío de paquetes, propias de las capas inferiores del modelo OSI.

Esta migración del plano de control, antes inherente a los dispositivos de red individuales, hace posible que la infraestructura de red sea abstraída como una entidad lógica, pudiendo estar a disposición de servicios y aplicaciones que gestionan la red de forma centralizada y como una única entidad virtual.

La gestión centralizada de la red y la separación de los planos de datos y de control, propias de las redes definidas por software, han abierto un nuevo modelo de negocio basado en la personalización de los servicios de red. Los servicios multimedia se han hecho imprescindibles en la vida cotidiana, abarcando una amplia gama de aplicaciones que han sido motivadas por las capacidades de interacción que estas tecnologías han propiciado. Esta necesidad de comunicación ha generado un severo incremento del tráfico de datos multimedia que circula por Internet, especialmente en los últimos años. Todo apunta a que esta tendencia creciente se mantendrá en los próximos años [Cis15].

En la actualidad, no es suficiente la entrega de un servicio multimedia si se desconoce la calidad de su prestación. Las demandas de los usuarios son cada vez mayores, al igual que las plataformas desde las que acceden a los servicios. Conceptos como la calidad de servicio (QoS) o la calidad de experiencia (QoE) han sido adoptados en diversas áreas de la tecnología, involucrando también a las tecnologías multimedia. Por la propia naturaleza de sus servicios, y en particular cuando se trata de transmisiones en directo, la medición de la calidad en tiempo real supone un reto importante que se debe afrontar con la tecnología

disponible. Los protocolos de comunicación modernos tienen en cuenta estos aspectos desde el punto de vista de la aplicación, sin contar con información del entorno de red que suele obtenerse de las capas inferiores del modelo OSI. Desde esta perspectiva, las redes definidas por software poseen los elementos suficientes para que las aplicaciones puedan conocer el estado de la red y puedan también controlar su comportamiento, haciendo uso de interfaces de programación que les otorguen la suficiente flexibilidad que hasta hoy no es posible encontrar en las arquitecturas de red tradicionales.

1.1. Objeto de la Investigación

El funcionamiento eficiente de los servicios multimedia sobre una red SDN depende de la capacidad de monitorizar su estado. Las soluciones típicas se concentran en estimar variables a nivel de red dentro del plano de datos (velocidad, pérdida, retraso) [VADK14]. El controlador puede hacer uso de esta información para reconfigurar la red y balancear la carga de datos entre diferentes usuarios. Sin embargo, el controlador desconoce cuál es la percepción de calidad de los usuarios con respecto a los datos multimedia transmitidos. Una deficiente gestión de la red disminuye la calidad de experiencia de usuario y al mismo tiempo incrementa la carga de procesamiento en los dispositivos terminales en su intento por mantener el nivel de calidad de los servicios, siendo necesaria la ejecución de algoritmos de alto nivel (por ejemplo, para la corrección de errores o verificación de integridad). Se sabe además que la corrección de deficiencias en capas inferiores reduce tareas de procesamiento en capas superiores.

En este trabajo se presenta el diseño de una arquitectura SDN para mejorar la calidad de experiencia de los usuarios con respecto a los servicios multimedia recibidos a través de una red definida por software. Para ello, se proporciona al controlador SDN información actualizada originada por los protocolos multimedia de alto nivel, que servirá para mejorar las decisiones de

encaminamiento de los paquetes en los conmutadores OpenFlow. La arquitectura propuesta permite a los dispositivos terminales (hosts, móviles, tabletas) intercambiar información en el plano de aplicación SDN. Además, se propone el protocolo PCMS (Protocolo de Control Multimedia en SDN) que proporciona al controlador información periódica actualizada y en tiempo real sobre la calidad de la transmisión multimedia.

Con la implementación de la arquitectura mediante agentes software, el protocolo PCMS y los módulos en el controlador de red, junto al uso de herramientas para el análisis de vídeo y tráfico de red, además de un entorno de red emulado, se comprobará la efectividad de la arquitectura en los experimentos desarrollados en el capítulo final.

1.2. Trabajos Relacionados

Trabajos recientes han extendido el alcance de las tareas para la gestión de tráfico a los dispositivos finales. En [SYF+15] se aprovechan las ventajas de las redes definidas por software para extender determinadas tareas de gestión, que tradicionalmente se realizan en servidores de seguridad (como *firewalls*), a los dispositivos terminales que se encargan, por ejemplo, de recopilar estadísticas de rendimiento de la red y comunicarlas por medio de interfaces de programación a otros nodos que procesan la información desde una perspectiva global. Se demuestra con ello que las tareas de gestión pueden ser programables (ya sea de forma centralizada o distribuida) en una red SDN.

La monitorización de redes es de mucha importancia para la obtención de parámetros que permitan determinar su estado. Dichos parámetros son dependientes de la información recopilada en el plano de datos. En [VADK14] se aborda la monitorización de redes SDN, desarrollando para ello un modelo centralizado que hace uso de Interfaces de Programación de Aplicaciones (*Application Program Interface* o API) para la obtención de parámetros de red del

plano de datos, su tratamiento y consolidación en el plano de control, y su comunicación con el plano de aplicación. De ese modo, se permite que otras entidades software puedan beneficiarse de la información procesada en el controlador, provistos de las interfaces apropiadas para el intercambio de datos (como REST API).

La evaluación de la calidad de vídeo en redes IP (*Internet Protocol*) se aborda en [NOALS12], donde se presenta un modelo basado en distintas métricas que se utilizan para evaluar la calidad de un vídeo. El análisis se realiza al finalizar la transmisión del contenido multimedia (*offline analysis*), enfocándose así en el resultado, más no en el proceso. Los informes de evaluación de la calidad se obtienen con el uso del software *Video Tester* desarrollado para ese fin. *Video Tester* posee componentes que distinguen las funciones de transmisión de contenido (implementadas mediante protocolos multimedia) de aquellas relacionadas con el análisis de datos y la evaluación de calidad antes referida.

La calidad de experiencia del usuario (QoE) [LCMP+13] es un concepto que ha tomado fuerza en los últimos años. QoE extiende el concepto tradicional de calidad de servicio considerando además el grado de aceptación de un usuario respecto a la percepción de un determinado servicio multimedia. Una de las principales áreas de análisis de la QoE es la entrega de contenido multimedia a través de mecanismos como *streaming*, *broadcast*, transmisión de archivos, etc. En ciertos casos (como la transmisión en directo), la medición de la calidad de experiencia en tiempo real se hace necesaria para evidenciar que un servicio multimedia cumple con las expectativas de calidad de un usuario.

En [PF14] y [KSKD+12] se exploran métodos de mejora de la calidad a través de la optimización de rutas. En ellos se definen ciertos elementos de control como parte de la arquitectura SDN, enfocados en la mejora de la calidad de experiencia multimedia basada en la optimización de rutas en una red definida por software. El primero, contrasta los resultados con métricas de medición de

la calidad. El segundo, plantea una arquitectura basada en la negociación del servicio y se trata de un trabajo en progreso que introduce los conceptos de arquitectura, calidad y asignación de rutas en una transmisión de datos multimedia.

1.3. Estructura del Trabajo

El presente trabajo está dividido en ocho capítulos. En el capítulo 2 se describen los conceptos subyacentes a una red definida por software y sus características.

El capítulo 3 explora los detalles de funcionamiento del protocolo OpenFlow y su relevancia en el desarrollo de la tecnología SDN. En particular, se describe la interacción entre los conmutadores y el controlador de red, a través del intercambio de mensajes OpenFlow y la correspondiente configuración de tablas de flujo en los dispositivos.

En el capítulo 4 se hace una revisión de los controladores SDN más importantes de la industria. Se analizan sus características más relevantes, principalmente desde la perspectiva de su arquitectura. Al finalizar se realiza una comparación entre los controladores examinados.

El capítulo 5 describe los protocolos RTSP, RTP y RTCP, utilizados para las transmisiones multimedia en tiempo real. Estos protocolos son de amplio uso en la actualidad y sirven de base para el despliegue de los experimentos realizados en este trabajo.

El capítulo 6 detalla la arquitectura propuesta para la mejora de la calidad de servicios multimedia. Se enumeran sus componentes software y el rol que desempeñan en los planos de control y de aplicación de la arquitectura SDN. Se presenta también el protocolo PCMS (Protocolo de Control Multimedia para SDN), que se ha propuesto para intercambiar mensajes entre los componentes de la arquitectura.

El capítulo 7 presenta los experimentos realizados para comprobar el funcionamiento de la arquitectura propuesta, así como los resultados obtenidos tras evaluar la calidad de experiencia del usuario mediante el uso de software especializado.

Finalmente, en el capítulo 8 se presentan las conclusiones del presente trabajo y se mencionan las posibles líneas de trabajo futuro que conduzcan a enriquecer la investigación realizada.

2. REDES DEFINIDAS POR SOFTWARE

2.1. Definición

Open Networking Foundation (ONF) [Ope12b] define una Red Definida por Software (SDN) como una arquitectura de red donde el control de la red es programable y está separada de las funciones de reenvío de paquetes, propias de las capas inferiores del modelo OSI.

En un contexto más extendido una Red Definida por Software es una arquitectura que optimiza y simplifica las operaciones de red, vinculando las aplicaciones con los dispositivos y servicios de red, sean éstos reales o virtuales. Para este cometido se emplea un punto de control centralizado a nivel lógico, denominado controlador SDN, que es el responsable de coordinar las comunicaciones entre las aplicaciones que deseen interactuar con los elementos de red. Para ello el controlador expone las funciones y operaciones de red a través de interfaces de programación mediante las cuales interactúa con las aplicaciones bidireccionalmente [NG13].

La Figura 2.1 muestra una vista lógica de la arquitectura SDN [Ope12b]. La arquitectura define tres planos: infraestructura, control y aplicación. En el plano de infraestructura, los conmutadores encaminan los paquetes según la configuración de sus tablas de flujo. En el plano de control residen los controladores SDN, que se encargan de configurar las reglas de encaminamiento en las tablas de flujo de los conmutadores. En el plano superior se encuentran las distintas aplicaciones SDN. Los planos se comunican por interfaces de programación que hacen posible su interacción. La inteligencia de la red está centralizada (a nivel lógico) en los controladores SDN, que son capaces de mantener una vista global de la red. Con SDN las organizaciones pueden controlar la red desde un punto central de administración y con

independencia del fabricante, simplificando así el diseño y operación de la red. SDN simplifica también los dispositivos de red en sí mismos, ya que éstos no necesitan soportar una gran cantidad de protocolos y estándares, sino limitarse a aceptar las instrucciones que llegan desde el controlador.

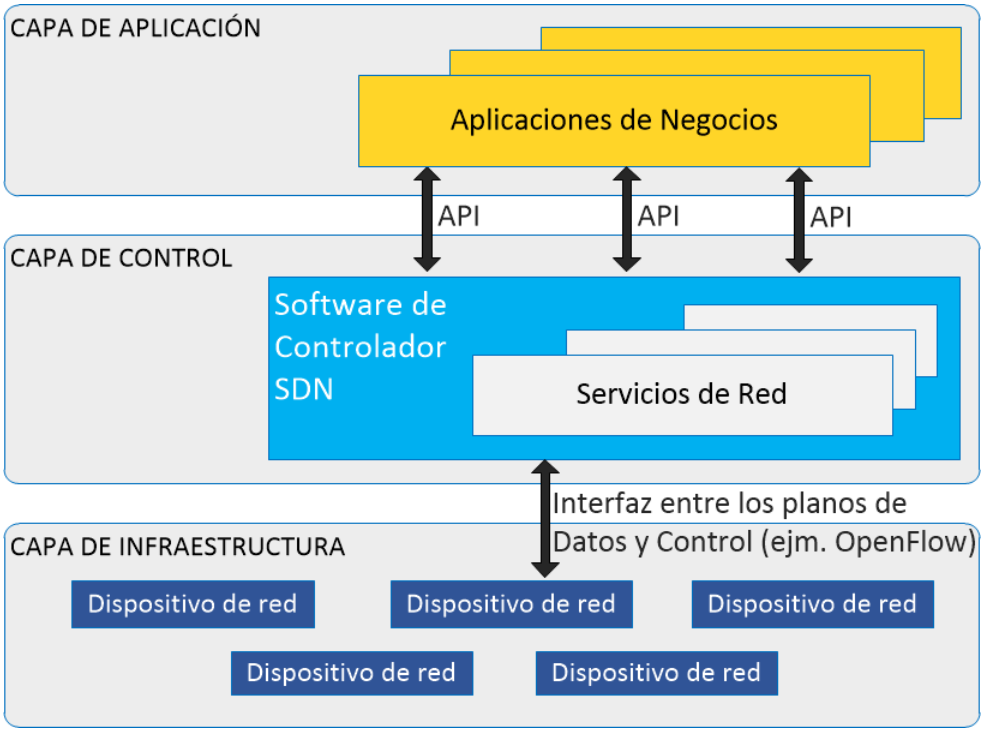

Figura 2.1: Arquitectura SDN.

La tecnología SDN ha sido influenciada por tecnologías predecesoras que a lo largo de la historia (en especial, durante los últimos veinte años) han dado lugar a nuevos paradigmas de gestión, introduciendo conceptos como el control centralizado, las redes programables, virtualización de las funciones de red, entre otros, que han sido relevantes para el desarrollo de esta tecnología emergente. A continuación se presenta una visión resumida de la historia de SDN, la separación de los planos de datos y de control, las aplicaciones SDN y el futuro de esta tecnología.

2.2. Historia de las Redes Definidas por Software

Las tecnologías predecesoras de SDN tienen sus orígenes en los años 80 cuando en las redes de telefonía de AT&T se introdujo el concepto de gestión centralizada de la red a través de NCP (*Network Control Point*) [Pri15b]. Disponer de un único punto de control para toda la red fue fundamental para dar soporte a un conjunto de aplicaciones de red que tenían como objetivo mejorar los servicios ofrecidos a los clientes. Mediante NCP se disponía de una visión global de la red telefónica que facilitó la asignación eficiente de circuitos de comunicación y la apertura de una amplia gama de nuevos servicios derivados de la capacidad de observar el comportamiento de la red en su conjunto. Por otra parte, el control centralizado hizo posible la evolución independiente de la infraestructura y de los servicios, según los requerimientos de datos, carga de trabajo, etc. La red se abstrae como una unidad lógica que se almacena en una base de datos que se consulta según los propósitos de cada aplicación; por ejemplo, para la localización eficiente de usuarios en la red de telefonía.

Otro hito importante en el desarrollo de SDN fue la aparición de las redes programables o activas o *Active Networking* [FRZ13], que consiste en crear redes de datos en las cuales los conmutadores otorgan un tratamiento personalizado a los paquetes mediante la ejecución de código. *Active Networking* se desarrolló a inicios de los años noventa por investigadores vinculados a DARPA (*Defense Advanced Research Projects Agency*). Las dificultades para integrar nuevas tecnologías en las redes de datos y el rendimiento poco eficiente en la ejecución de ciertas tareas redundantes en distintos protocolos fue la motivación para la creación de este tipo de redes. Asimismo, se pretendía contar con un entorno de red que acelere la innovación sin tener que esperar años desde que una propuesta tecnológica se proponga hasta que se estandarice y despliegue. Se propusieron dos modelos de programación: el primero, llamado Modelo de Cápsula, plantea que el código a ejecutar en los nodos viaja encapsulado en

paquetes de datos; es un modelo de administración en banda que envía el tráfico de gestión a través de la propia red. El segundo, llamado Modelo de Encaminador Programable, plantea que el código a ejecutar en los nodos se proporciona mediante un esquema fuera de banda (*out-of-band*), independizando el tráfico de gestión del tráfico de la red IP.

En *Active Networking*, los encaminadores activos (*active routers*) coexisten con los encaminadores IP y, a diferencia de éstos, realizan tareas de procesamiento adicionales cada vez que encuentran código que ejecutar. En consecuencia, el comportamiento de la red se hace personalizable, con la flexibilidad que otorga un lenguaje de programación. No obstante, y siendo notables las ventajas de este paradigma, no se extendió a otros dominios (aparte del científico) al no existir en su momento una aplicabilidad inmediata que justifique una demanda existente, como la que existe hoy, por ejemplo, con los centros de datos. Otra limitación de *Active Networking* fue la incompatibilidad de los encaminadores activos con el hardware existente, siendo necesario el uso de hardware específico. Otros aspectos, como la seguridad en la ejecución del código, no fueron suficientemente cubiertos. A pesar de las limitaciones se reconocen las contribuciones de esta tecnología a las Redes Definidas por Software, principalmente en la definición de funciones programables en la red que personalicen su comportamiento y faciliten la innovación.

La virtualización de redes es también un hito importante en el desarrollo de SDN. Se entiende por virtualización de redes a la representación de una o más topologías de red lógicas en la misma infraestructura física. Las redes de área local virtuales (VLAN) o la gestión de redes virtuales de VMware son ejemplos de este tipo de tecnología. Las redes virtuales permiten la existencia de múltiples encaminadores lógicos en una misma plataforma física, otorgando beneficios como el aislamiento de recursos de procesamiento, memoria, ancho de banda, tablas de encaminamiento, etc. Del mismo modo, la virtualización de redes permite alcanzar niveles superiores de personalización del software de

encaminamiento y flexibilidad en la asignación de recursos de procesamiento. Algunos ejemplos de redes virtuales son Tempest, VINI y Cabo.

En el campo de la virtualización, las contribuciones que introdujo VMware hace más de diez años fueron fundamentales para que esta tecnología se extendiera también a los entornos de red a cualquier escala, desde grandes centros de datos hasta redes corporativas. A las contribuciones de VMware se sumaron nuevas necesidades motivadas por los cambios en los modelos computacionales. El caso más significativo es la aparición de la computación en la nube y la consiguiente necesidad de compartir una misma infraestructura física entre distintos clientes (o usuarios) de los centros de datos. Es preciso mencionar que la virtualización de redes no es un requisito indispensable para el funcionamiento de una red definida por software; lo inverso tampoco es aplicable. Sin embargo, son tecnologías que han constituido una simbiosis destacable.

2.3. Separación de Planos

Se ha mencionado que la separación de planos es una característica distintiva en las redes definidas por software. Se explica a continuación el rol que cada uno desempeña.

2.3.1. El Plano de Control

El plano de control es el componente que establece la lógica y el conjunto de datos que se utilizan para controlar el encaminamiento en una red SDN. Los protocolos de red o las reglas de configuración de los cortafuegos son ejemplos de ello [Pri15a].

El objetivo del plano de control es administrar las tablas de flujo en los conmutadores de la red, lo cual se realiza a través de la inserción, eliminación o modificación de entradas en las tablas de flujo para definir reglas de

encaminamiento. Para lograr dicho objetivo, el plano de control debe abstraer la red desde un punto de vista global que satisfaga ciertas restricciones. Esta abstracción de la red es programable y se logra mediante un Sistema Operativo de Red (NOS) valiéndose de algún protocolo que permita obtener información del plano de datos a través de una Interfaz de Programación de Aplicaciones (*Southbound API*). Con esa información se define el comportamiento de la red, que se traduce en reglas de encaminamiento que el controlador SDN configura en los conmutadores. En el capítulo cuatro se exploran los controladores SDN más comunes y sus características programables.

2.3.2. El Plano de Datos

El plano de datos se encarga de procesar los datagramas que ingresan a través de los medios físicos (cable, fibra, medios de comunicación inalámbrica) a través de una serie de operaciones en la capa de enlace (después de ensamblar el datagrama realizan una comprobación de errores). Los datagramas bien formados se procesan en el plano de datos mediante búsquedas en las tablas de encaminamiento que han sido programadas previamente por el plano de control. La única excepción a este flujo de procesamiento se da cuando el paquete no corresponde con ninguna de las reglas establecidas; por ejemplo, cuando se detecta un paquete con destino desconocido, se envía al controlador de red que podrá procesarlo según la lógica establecida en las aplicaciones programadas [NG13]. Es importante señalar que el plano de datos puede residir en elementos software, hardware de diverso tipo (GPU, CPU, FPGA, ASIC), o en una combinación de ambos. El reenvío de paquetes IP y las funciones de conmutación de la capa de enlace son funciones propias del plano de datos [Pri15a].

2.4. Ventajas de la Separación de Planos

En primer lugar, la separación de planos permite que cada uno se desarrolle independientemente. El plano de control está ligado a elementos software que pueden evolucionar sin estar condicionados a una plataforma hardware específica.

En segundo lugar, es posible controlar la red con software de alto nivel, implementando en ella todas las buenas prácticas de desarrollo del software. La corrección de errores, la depuración, el control de la calidad, etc., se ajustan a los patrones ya establecidos en el ciclo de vida del software.

El coste es otro aspecto a considerar. El procesamiento de datos en los equipos de red (conmutadores, encaminadores) es muy alto comparado con el coste que supone la ejecución de programas en hardware de propósito general. No menos importante es el coste de adquisición que determinan los fabricantes, que está ligado al hardware y al coste de propiedad intelectual que requiere, en ocasiones, el pago de licencias de uso. Aunque en SDN no se suprimen los costes, las posibilidades son mayores porque están abiertas a nuevas posibilidades de despliegue.

Gracias a la separación de planos es posible conseguir también una mayor estabilidad de la red, al requerirse una menor carga de procesamiento en los dispositivos de red y tablas de encaminamiento más pequeñas. En el esquema SDN los sistemas de los nodos intermedios son menos propensos a peligrar su operatividad por la presencia de errores en el software, fallos en la actualización del *firmware* (por ejemplo, actualizaciones de seguridad), etc.

2.5. Ejemplos de la Separación de Planos

2.5.1. RCP (Routing Control Platform)

La Plataforma de Control de Encaminamiento (RCP) [CCF+05] es un ejemplo de separación de los planos de datos y de control. Fue creada en 2004 ante la necesidad de mejorar la administración de rutas BGP en un sistema autónomo (AS). RCP define una arquitectura que concentra las funciones de encaminamiento en un servidor central (RCS) por cada sistema autónomo y se apoya en dos módulos: el motor BGP y el visor IGP. Los elementos de la arquitectura RCP se muestran en la Figura 2.2.

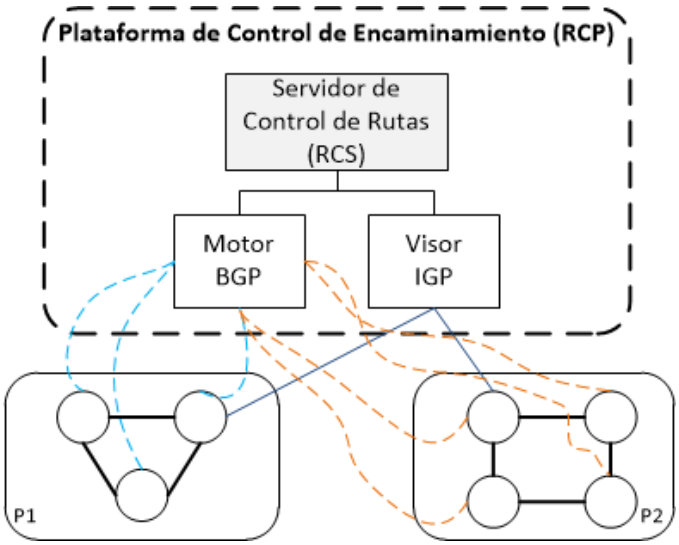

Figura 2.2: Arquitectura RCP.

El RCS se encarga de realizar el cálculo de las rutas BGP para todo el sistema autónomo. Con la información proporcionada por el motor BGP elabora un esquema de encaminamiento manteniendo una visión global de la red gracias a la información de la topología que le facilita el visor IGP. Las rutas calculadas se asignan a los encaminadores de la red. RCP busca resolver el problema del protocolo BGP que basa su funcionamiento en decisiones locales en lugar de globales.

2.5.2. Ethane

Ethane [CFP+07] constituye una arquitectura que introduce el concepto de controlador de dominio en una red cuyos planos de control y de datos se encuentran separados. Ethane data de 2007 y permite que los administradores de red definan un único conjunto de políticas de red que después se asigna a los conmutadores. Las políticas son configurables a través de software y definen los flujos de comunicación, así como las condiciones que deben cumplir los paquetes que circulan por la red. En ese esquema, el controlador, manteniendo una visión global de la red, asume el control de las funciones más complejas, incluyendo el encaminamiento, el direccionamiento, la asignación de nombres y la gestión de la seguridad. En la Figura 2.3 se muestra un ejemplo de comunicación entre dos nodos a través de la red Ethane.

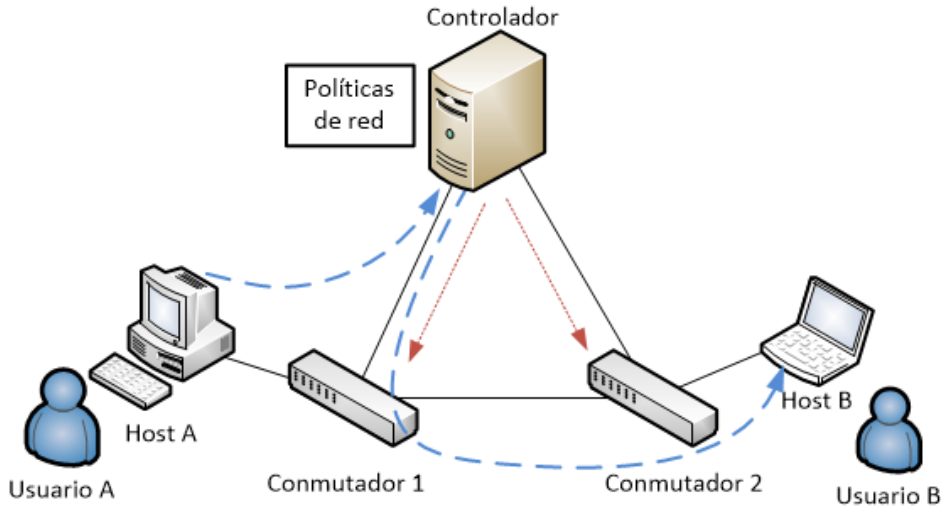

Figura 2.3: Ejemplo de comunicación en una red Ethane.

La comunicación se realiza en cinco pasos. En primer lugar, se registran los nodos con las credenciales necesarias para su autenticación. A continuación, los conmutadores establecen conexión con el controlador para que éste construya la topología de red. Con la topología establecida, se realiza el proceso de autenticación de *hosts*. Una vez autenticados, el controlador realiza la configuración de flujos en los conmutadores según las políticas definidas con

anterioridad. Finalmente, el controlador de dominio permite o deniega el flujo de comunicación entre los participantes de la comunicación (A y B en el ejemplo). La principal limitación para la adopción masiva de Ethane fue su escaso soporte en el hardware de red.

2.5.3. OpenFlow

El protocolo OpenFlow [MAB+08] ha sido determinante para la creación de las redes definidas por software. Se plantea como un estándar abierto que sirve como interfaz de comunicación entre los planos de datos y de control, permitiendo que el controlador pueda acceder y manipular los flujos de comunicación en el plano de datos mediante la administración de tablas de flujo (*flow tables*) en los conmutadores. Basado en Ethane, fue creado en 2008 en la Universidad de Stanford como resultado de los esfuerzos promovidos por la comunidad científica y empresarial. En OpenFlow es fundamental separar los planos de datos y de control. En la actualidad, la ONF se encarga de mantener el estándar. El protocolo OpenFlow se trata detalladamente en el siguiente capítulo.

2.6. Áreas de Aplicación de SDN

El despliegue de una red definida por software en un centro de datos hace posible llevar a cabo tareas de gestión que sólo pueden abstraerse desde un enfoque centralizado. El consumo de energía es crítico en su administración [NMN+14] y está directamente relacionado con la carga de trabajo presente en cada elemento de procesamiento. Una solución como ElasticTree [HSM+10], basada en SDN y con una visión global de la red, puede determinar los requerimientos mínimos para satisfacer las condiciones de tráfico requeridas en un proceso de comunicación, apagando los conmutadores que no sean necesarios y reduciendo así el consumo de energía gracias a una eficiente gestión de la red. En los centros de datos también son críticas las funciones de

migración de máquinas virtuales y su correspondiente direccionamiento, que pueden ser automatizados por software, evitando que se interrumpan los procesos de comunicación o eliminando la necesidad de reconfigurar los equipos migrados por una correcta gestión del encaminamiento en las capas de enlace de datos y de red.

Las soluciones de encaminamiento y monitorización centralizada de la red [KF13] son construidas sobre una arquitectura SDN. La configuración de políticas globales que respondan a eventos de red en capas inferiores no es posible de implantar en los modelos de red tradicionales, pero sí es factible de definir desde las aplicaciones cuando los planos de datos y de control se encuentran separados, como en el caso de SDN. Soluciones de este tipo son aplicables a entornos pequeños (*home networking*), redes empresariales y de investigación.

En el campo de las redes móviles [VCBPBLGV14] el paradigma SDN puede aportar importantes beneficios. La mayoría de proveedores de soluciones móviles comparten necesidades similares que pretenden ser resueltas a través de arquitecturas como SDMN (*Software Defined Mobile Network*), propuesta para facilitar la innovación y creación de redes programables a través de estándares abiertos. El modelo SDMN se compone de dos elementos: un motor para el reenvío de paquetes basado en flujos móviles (*MobileFlow Forwarding Engine* o MFFE), y un controlador que gestiona los flujos móviles (*MobileFlow Controller* o MFC). MFFE se corresponde con el plano de datos y tiene una estructura más compleja que OpenFlow, mientras que MFC representa un plano de control de alto rendimiento, en el cual se desarrollan las aplicaciones para redes móviles.

Las aplicaciones multimedia requieren niveles altos de disponibilidad y eficiencia de la infraestructura de red. Según un estudio de Cisco [Cis15], la suma del tráfico IP de paquetes de vídeo en todas sus variantes alcanzará más del 80% del tráfico global en el año 2019. Se ha mencionado también la

importancia de conceptos como la calidad de servicio (QoS) y la calidad de experiencia de usuario (QoE), que han adoptado una creciente importancia. En ese contexto, SDN ofrece funcionalidades para la optimización de la infraestructura de red.

La tecnología SDN es aplicable también a entornos de red WAN. Un caso de ejemplo es SDX (*SDN-Based Exchange*) [GVS+14], que consiste en la implementación de Puntos de Intercambio de Internet (*Internet Exchange Point* o IXP) que utilizan SDN para el cálculo de rutas entre redes administradas por distintos proveedores de servicios de Internet (*Internet Service Providers* o ISP). En SDX, el servidor central de rutas se encarga de establecer sesiones BGP (*Border Gateway Protocol*) con los enrutadores de los sistemas autónomos, para obtener y consolidar información global de rutas. Con esa información el controlador SDX define reglas de encaminamiento que posteriormente configura en los conmutadores SDX, aplicando para ello criterios de balanceo de carga, administración del tráfico entrante, filtrado, etc.

2.7. Desafíos de la Tecnología SDN

La escalabilidad es un factor importante, y uno de los constantes desafíos de las redes definidas por software [SSHC+13], puesto que no se conoce a priori hasta qué límite pueda ser necesario extender una red de comunicación. En las arquitecturas de red tradicionales, esto lleva consigo algunos aspectos como la limitación del hardware en cuanto a sus capacidades de procesamiento, memoria, restricciones del fabricante (software propietario, estándares soportados, etc.), número de entradas soportadas en las tablas de encaminamiento, algoritmos de búsqueda, aspectos de rendimiento, entre muchos otros que restringen las opciones de escalar cuando las necesidades lo demandan. SDN hace frente a estas limitaciones con la separación de planos y el uso de estándares de comunicación abiertos para distintas plataformas de hardware.

Otro de los desafíos importantes es la seguridad [SSHC+13]. La arquitectura SDN lleva consigo el riesgo de enfrentarse a un fallo del controlador, o que se vea comprometida su operatividad. El controlador SDN está lógicamente centralizado y físicamente distribuido, con los riesgos de operación propios de ese modelo [LWH+12]. Aunque es un aspecto que viene siendo adaptado a las necesidades actuales, como por ejemplo el diseño de controladores SDN elásticos (es decir, que se extiendan a otros nodos computacionales bajo demanda) [DHM+13], la dependencia de un elemento que centralice la lógica de operación de una red es el aspecto que requiere de mayor cuidado en todos los despliegues SDN presentes y futuros.

La interoperabilidad de tecnologías SDN es un desafío permanente que será clave para la transición del modelo tradicional al enfoque SDN [SSHC+13]. Para que se pueda hacer viable una transición será necesario brindar un soporte continuado a las tecnologías de red existentes. Sólo así los procesos de migración podrán ser progresivos, garantizando la existencia de entornos híbridos. Un protocolo importante para este fin es el PCEP (*Path Computation Element Communication Protocol*).

SDN abre nuevas oportunidades para las redes empresariales, que pueden implantar nuevos servicios en formas que antes eran imposibles de concebir. La gestión de la seguridad, la monitorización, la configuración automatizada de equipos, la gestión de cambios y actualizaciones desde un enfoque centralizado son retos que la tecnología SDN ha conseguido afrontar.

Es indudable que SDN marca un nuevo paradigma en la gestión de redes, que se hará más evidente a medida que se implementen soluciones de red modernas, escalables, seguras, compatibles e interoperables. SDN ha emergido teniendo en cuenta estos principios y las limitaciones de las arquitecturas de red existentes hasta hoy. El respaldo del sector científico y de la industria es imprescindible para asegurar su evolución.

3. EL PROTOCOLO OPENFLOW

3.1. Introducción

El concepto de redes programables se ha planteado como una forma de facilitar la evolución de las redes de datos. Se ha explicado en el capítulo anterior que la característica distintiva en las redes programables, y en SDN en particular, es el desacoplo de las decisiones de control de las funciones de reenvío de paquetes en lo que se denominan planos de control y de datos. Esta separación proporciona abstracción de la red flexible y programable. Además de la abstracción de redes, la arquitectura SDN proporciona un conjunto de Interfaces de Programación de Aplicaciones que simplifican la implementación de servicios de red comunes; por ejemplo, el encaminamiento, la multidifusión, la gestión de la seguridad, el control de accesos, la gestión de ancho de banda, entre otras. En SDN, la inteligencia de la red está lógicamente centralizada en controladores basados en software, mientras que los conmutadores se convierten en simples dispositivos de reenvío de paquetes, cuyo comportamiento es programado desde el controlador a través de una interfaz abierta. Una de esas interfaces, la más conocida, es OpenFlow [Azo13].

La motivación para crear OpenFlow surgió de la necesidad de contar con una red experimental para la investigación que opere en paralelo con la infraestructura de red propia de un campus [MAB+08]. El objetivo fue conseguir que los fabricantes incorporen una plataforma abierta en sus dispositivos, y que sea accesible desde una interfaz que permita configurar su comportamiento, pero que a la vez mantenga compatibilidad con las plataformas existentes (propias del fabricante). Se buscó además que la arquitectura esté disponible en entornos de alto rendimiento y en dispositivos de bajo coste.

En 2011 un grupo de operadores de red, proveedores de servicios y representantes de la industria crearon la *Open Networking Foundation* (ONF) [Op1], una organización que promueve la difusión de SDN en la industria y se encarga de estandarizar el protocolo OpenFlow. En la actualidad, la versión más reciente es la 1.5, creada en diciembre de 2014. Aunque el estándar sigue evolucionando, la versión que ampliamente se ha desplegado e implementado es la 1.0.0. Se explicarán los principios de este estándar en las secciones siguientes.

3.2. Conmutador OpenFlow

Un conmutador OpenFlow tiene los siguientes componentes: una tabla de flujo y un canal seguro de comunicación. La tabla de flujo sirve para realizar las búsquedas de paquetes de acuerdo a ciertas reglas definidas que determinan los criterios para su encaminamiento. El canal seguro sirve para comunicarse con el controlador, el cual se encarga de configurar el conmutador a través del protocolo OpenFlow [Ope09]. En la Figura 3.1 se muestra un esquema del conmutador y de sus elementos.

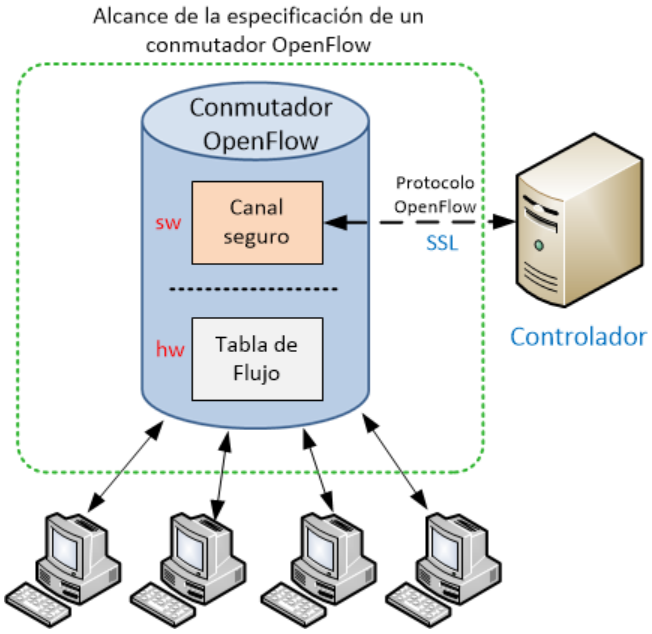

Figura 3.1: Componentes de un conmutador OpenFlow.

En la versión 1.3 del protocolo OpenFlow [Ope12a] se introduce un tercer componente denominado grupo de tablas (*group table*), capaz de almacenar una o más tablas de flujo.

3.3. Tablas de Flujo

Una tabla de flujo es una colección de entradas con la estructura que se muestra en la Tabla 3.1.

Campos de encabezado	Contadores	Acciones

Tabla 3.1: Estructura de una entrada en una tabla de flujo.

Los elementos de la entrada son:

- Campos de encabezado: Sirven para definir criterios de correspondencia (*match*) con los paquetes.
- Contadores: Se actualizan cada vez que se produce una correspondencia.
- Acciones: Instrucciones a realizar cuando se produzca una correspondencia.

3.3.1. Campos de Encabezado

En la Tabla 3.2 se mencionan los campos de un paquete que pueden ser usados para definir criterios de correspondencia.

Ingress Port	Ethernet Source	Ethernet Destination	Ethernet Type	VLAN ID	VLAN priority	IP src	IP dst	IP protocol	IP ToS bits	TCP/ UDP src. port	TCP/ UDP dst. port

Tabla 3.2: Campos de un paquete utilizados para crear criterios de correspondencia.

En la Tabla 3.3 se describen las propiedades de cada campo:

Campo	Bits	Casos aplicables	Notas
Ingress Port	(Dependiente de la implementación)	Todos los paquetes	Representación numérica del puerto de entrada, empezando en 1
Ethernet Source Address	48	Todos los paquetes en los puertos habilitados	
Ethernet Destination Address	48	Todos los paquetes en los puertos habilitados	
Ethernet Type	16	Todos los paquetes en los puertos habilitados	
VLAN Id	12	Todos los paquetes Ethernet del tipo 0x8100	
VLAN Priority	3	Todos los paquetes Ethernet del tipo 0x8100	Campo PCP (*Priority Code Point*) de una VLAN.
IP Source Address	32	Todos los paquetes IP y ARP	Se le puede aplicar una máscara de subred
IP Destination address	32	Todos los paquetes IP y ARP	Se le puede aplicar una máscara de subred
IP Protocol	8	Todo paquete IP e IP sobre Ethernet, paquetes ARP	Sólo los 8 bits menos significativos del código ARP son utilizados
IP ToS Bits	6	Todos los paquetes IP	
Transport Source Port / ICMP Type	16	Todos los paquetes TCP, UDP e ICMP	Sólo son utilizados los 8 bits menos significativos del tipo ICMP.
Transport Destination Port / ICMP Code	16	Todos los paquetes TCP, UDP e ICMP	Sólo son utilizados los 8 bits menos significativos del tipo ICMP.

Tabla 3.3: Longitudes de campo y la forma en que son aplicados en las entradas de flujo.

3.3.2. Contadores

En OpenFlow los contadores se mantienen por tabla, por flujo, por puerto y por cola. La Tabla 3.4 muestra el conjunto de contadores requeridos por el protocolo y la longitud de cada uno.

Contador	Bits
Por Tabla	
Entradas Activas	32
Búsquedas de Paquetes	64
Correspondencia de Paquetes	64
Por Flujo	
Paquetes Recibidos	64
Bytes Recibidos	64
Duración (segundos)	32
Duración (nanosegundos)	32
Por Puerto	
Paquetes Recibidos	64
Paquetes Transmitidos	64
Bytes Recibidos	64
Bytes Transmitidos	64
Paquetes eliminados en recepción	64
Paquetes eliminados en transmisión	64
Errores de recepción	64
Errores de transmisión	64
Errores de alineamiento en las tramas recibidas	64
Errores de sobre procesamiento de recepción	64
Errores de CRC en recepción	64
Colisiones	64
Por Cola	
Paquetes Transmitidos	64
Bytes Transmitidos	64
Errores sobre Ejecución de Transmisión	64

Tabla 3.4: Lista de contadores requeridos para su uso en mensajes de estadísticas.

3.3.3. Acciones

Cada entrada en la tabla de flujo está asociada con una o muchas acciones que determinan la forma en que el conmutador debe encaminar el paquete. Si no se especifica acción alguna en la entrada de flujo, se descarta el paquete. Si, por el contrario, no existe ninguna entrada en la tabla de flujo que corresponda con el paquete, éste se envía al controlador para que determine la forma en que debe tratarse. Las posibles acciones de un conmutador OpenFlow se clasifican en obligatorias y opcionales, y son descritas a continuación.

3.3.3.1. Acciones Obligatorias

Las acciones obligatorias deben estar soportadas en todo conmutador, ya sea puramente OpenFlow (*OpenFlow Only*), o compatible con OpenFlow (*OpenFlow-enabled*), y son las siguientes:

- **Forward:** Los conmutadores OpenFlow deben soportar el reenvío de paquetes a puertos físicos y a los siguientes puertos virtuales:
    - *ALL:* Envía un paquete de salida a todos los puertos, excluyendo el puerto de entrada.
    - *CONTROLLER:* Encapsula el paquete y lo envía al controlador.
    - *LOCAL:* Envía el paquete a la pila de procesamiento local del conmutador.
    - *TABLE:* Realiza acciones en la tabla de flujo (sólo para paquetes del tipo *Packet-Out*).
    - *IN PORT:* Envía el paquete de salida por el puerto de entrada.
- **Drop:** Toda entrada en la tabla de flujo que no especifica acción indica que los paquetes coincidentes deben ser descartados.

3.3.3.2. Acciones Opcionales

Las acciones opcionales son las siguientes:

- **Forward:** Los conmutadores OpenFlow podrían soportar los siguientes puertos virtuales:
    - *NORMAL:* Procesa el paquete utilizando el esquema de encaminamiento tradicional soportado por el conmutador (procesamiento tradicional de capas 2 y 3).

- *FLOOD:* Difunde el paquete a lo largo del mínimo árbol de expansión (*spanning tree*), sin incluir la interfaz de entrada.

- **Enqueue:** Reenvía el paquete a través de una cola perteneciente a un puerto.

- **Modify-Field:** Opción de modificar los valores de los encabezados de un paquete, como por ejemplo, establecer el identificador de VLAN, prioridad, dirección IP de destino, etc.

3.4. Flujo de Correspondencia (*Matching*)

El flujo de acciones que se realizan para establecer correspondencia entre un paquete y una entrada en la tabla de flujo se muestra en la Figura 3.2.

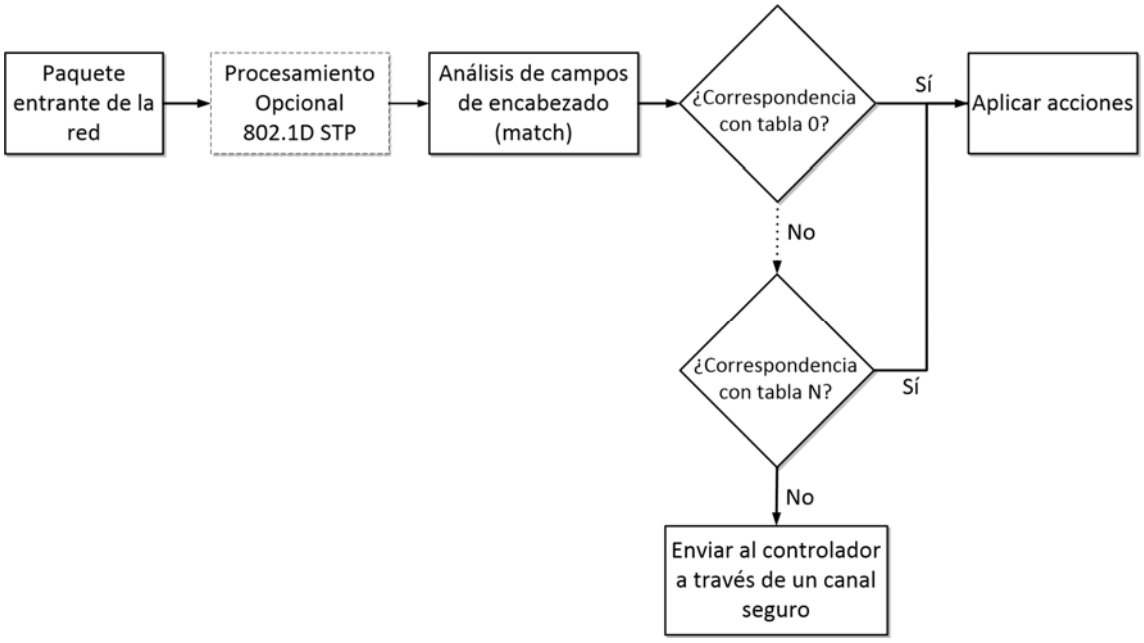

Figura 3.2: Flujo de un paquete en un conmutador OpenFlow v1.0.0.

El encabezado de un paquete se examina en cada entrada de las tablas de flujo para encontrar correspondencia con algún criterio. Si el paquete no corresponde con ninguna entrada se envía al controlador. En caso contrario, se ejecutan las acciones indicadas.

En la versión 1.3 del protocolo se introdujeron mejoras como el soporte de conjuntos de acciones (*Action Sets*) y grupos de tablas. Cada tabla puede actualizar campos, modificar los conjuntos de acciones, etc., según se defina en cada caso.

3.5. Canal Seguro

El canal seguro es la interfaz que conecta cada conmutador OpenFlow con el controlador. A través de esta interfaz, el controlador configura y administra el conmutador, recibe eventos del conmutador y envía paquetes fuera del conmutador.

3.5.1. Tipos de Mensajes OpenFlow

El protocolo soporta tres tipos de mensajes: controlador a conmutador (*controller-to-switch*), asíncronos (*asynchronous*) y simétricos (*symmetric*), cada uno con un conjunto de subtipos.

3.5.1.1. Controlador a Conmutador

Estos mensajes son iniciados por el controlador y pueden requerir o no una respuesta desde el conmutador.

- **Features:** A través de una sesión TLS (*Transport Layer Security*) el controlador solicita al conmutador las funcionalidades soportadas. El conmutador debe responder con un mensaje que especifique las capacidades que soporta.
- **Configuration:** El controlador puede establecer y consultar parámetros de configuración del conmutador. El conmutador sólo responde a las consultas del controlador.

- **Modify-State:** Estos mensajes son enviados por el controlador para administrar el estado de los conmutadores. Su propósito primario es añadir, eliminar o modificar las tablas de flujo, y establecer las propiedades del puerto.

- **Read-State:** Estos mensajes son usados por el controlador para obtener estadísticas de las tablas de flujo de los conmutadores, puertos y entradas de flujo individuales.

- **Send-Packet:** Son usados por el controlador para enviar paquetes de salida a través de un puerto específico del conmutador.

- **Barrier:** Estos mensajes de solicitud y respuesta son usados por el controlador para asegurarse que las dependencias se han cumplido o para recibir notificaciones de operaciones completadas.

3.5.1.2. Asíncronos

Los mensajes asíncronos son iniciados por el conmutador sin que el controlador los solicite. Los conmutadores envían estos mensajes al controlador para notificar la llegada de un paquete, un cambio de estado en el conmutador o un error. Los cuatro subtipos principales son:

- **Packet-in:** Se usa para todos los paquetes que no corresponden con ninguna entrada en la tabla de flujo, o para aquellos que tengan configurada como acción el envío al controlador.

- **Flow-Removed:** Cada vez que se añade una entrada a la tabla de flujo a través de un mensaje *Modify-State*, se especifica un tiempo de inactividad (*idle timeout*) después del cuál debe ser eliminada la entrada en la tabla, y un tiempo de vida (*hard timeout*) que elimine la entrada después de un tiempo determinado, independientemente de la actividad. En estos mensajes de

modificación también se especifica si el conmutador debe o no enviar un mensaje *Flow-Removed* al controlador cada vez que expira la entrada en la tabla de flujo.

- **Port-status:** Estos mensajes son enviados por el conmutador cada vez que cambia el estado en la configuración de un puerto.

- **Error:** Son usados para notificar al controlador problemas en el conmutador.

3.5.1.3. Simétricos

Los mensajes simétricos son iniciados, bien por el conmutador bien por el controlador, enviándose sin necesidad de solicitud previa. Existen tres subtipos:

- **Hello:** Son usados por el conmutador y controlador durante el establecimiento de la conexión.

- **Echo:** Los mensajes de solicitud (*echo request*) requieren siempre una respuesta (*echo reply*). Pueden ser generados por el conmutador o controlador y son usados para obtener la latencia, ancho de banda o estado de una conexión.

- **Vendor:** Son usados para ofrecer funcionalidades adicionales dentro del posible espacio de mensajes OpenFlow. Está sujeto a futuras revisiones.

3.5.2. Características de la Conexión Segura

Se ha mencionado antes que la conexión se hace entre conmutador y controlador a través de una sesión TLS. El puerto TCP que se utiliza por defecto es el 6633. El conmutador y controlador se autentican mediante el intercambio de certificados digitales.

En caso de pérdida de conexión con el controlador, el conmutador trata de conectarse con los controladores de respaldo que tenga configurados. Si los intentos por conectarse con un controlador resultan infructuosos, el conmutador pasa a un estado de emergencia (*emergency mode*). En este estado, todas las entradas de la tabla de flujo se eliminan y el comportamiento del conmutador está determinado sólo por las entradas que tengan activado el bit de emergencia cuando fueron añadidas a la tabla.

Los conmutadores OpenFlow pueden soportar, opcionalmente, el protocolo IEEE 802.1D (*Spanning Tree Protocol)*. Los conmutadores que soporten dicho protocolo deberán procesar los paquetes 802.1D antes de realizar búsquedas en la tabla de flujo.

4. CONTROLADORES SDN

4.1. Introducción

Los aspectos distintivos de una red definida por software son la separación de planos, la capacidad de hacer de las redes elementos programables y la gestión centralizada de la red desde un controlador. El plano de control puede seguir un modelo centralizado o distribuido. En este último caso, cada instancia del controlador mantiene las tres características citadas al inicio, desde el contexto que le corresponde según la configuración de la red. En las siguientes secciones se presenta un modelo de controlador idealizado que trata de resumir las características y funcionalidades comunes a los controladores de red, luego se examinan algunos controladores ofrecidos por ciertos proveedores de la industria junto a otros proporcionados por la comunidad *open source*.

4.2. Modelo de Controlador Idealizado

El controlador de red, llamado también Sistema Operativo de Red (*Network Operating System* o NOS) es el programa que permite construir y ejecutar aplicaciones que controlan el comportamiento de una red definida por software. Desde el punto de vista de las aplicaciones, es el elemento que ofrece una visión abstracta y global de la red. Desde el punto de vista de la red, es el elemento que traduce las instrucciones de las aplicaciones en instrucciones de configuración para definir el comportamiento de los elementos de red.

En [NG13] se ofrece un modelo de controlador idealizado que permite ampliar su descripción, señalando las funciones que desempeñan sus principales componentes y su relación con las aplicaciones y elementos de red. En la Figura 4.1 se muestra una vista esquemática del modelo.

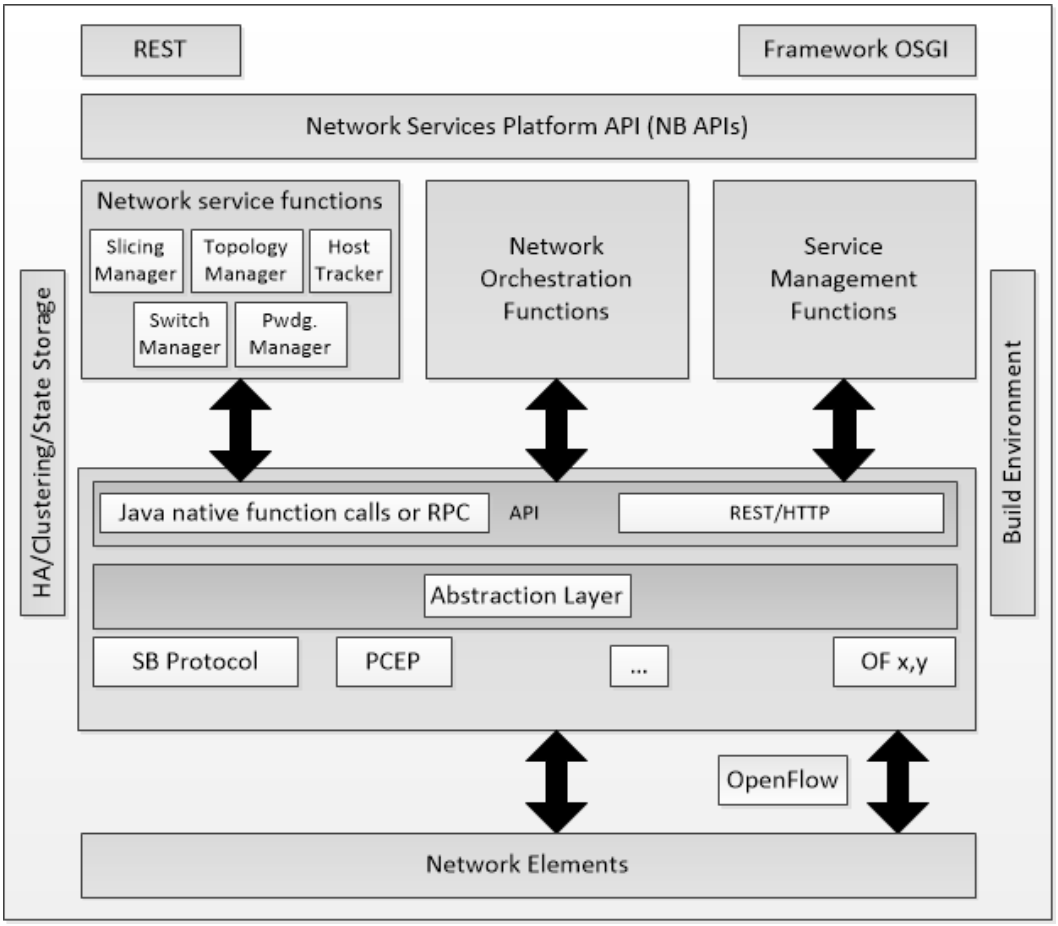

Figura 4.1: Modelo de controlador idealizado.

En un sentido amplio, un controlador SDN es una colección de sistemas que, en conjunto, proporcionan:

- Administración del estado de la red y, en algunos casos, el manejo y la distribución de este estado, involucrando una base de datos. Dicha base de datos sirve como repositorio para la información obtenida desde los dispositivos de red y desde las aplicaciones SDN, incluyendo el estado de la red, información de configuración, topología e información de control de las sesiones.

- Un modelo de datos de alto nivel que capture la relación entre recursos administrados, políticas y otros servicios proporcionados por el controlador. En muchos casos, son construidos utilizando el lenguaje de modelado Yang.

- Una API moderna, comúnmente de tipo REST (*Representational State Transfer*), que presente los servicios del controlador a las aplicaciones. Esta interfaz facilita la mayoría de opciones de interacción entre aplicación y controlador.

- Un canal TCP seguro para la conexión entre el controlador y los agentes en los dispositivos de red.

- Mecanismos para la búsqueda y detección de dispositivos, topología y servicios; un sistema de cálculo de rutas, y otros servicios centralizados que brinden información de la red y sus dispositivos asociados.

4.3. Controladores de Red en la Industria

Se examinan algunos de los principales controladores SDN, principalmente los de código abierto.

4.3.1. Floodlight

Floodlight [Fdl] es un controlador SDN de código abierto. Es una contribución de Big Switch Networks [Bsn] a la comunidad. Cuenta con licencia Apache, está escrito en Java y soporta el protocolo OpenFlow como interfaz de comunicación (*southbound interface*) con los elementos de red.

La arquitectura de Floodlight es modular, con componentes que incluyen las funcionalidades descritas en el modelo de controlador idealizado de la sección anterior. Floodlight adicionalmente proporciona un conjunto de aplicaciones (o módulos) en Java integradas en el controlador [Fdl]. La Figura 4.2 muestra la relación entre el controlador, las aplicaciones integradas que presentan interfaces de programación en Java y las aplicaciones accesibles desde interfaces REST.

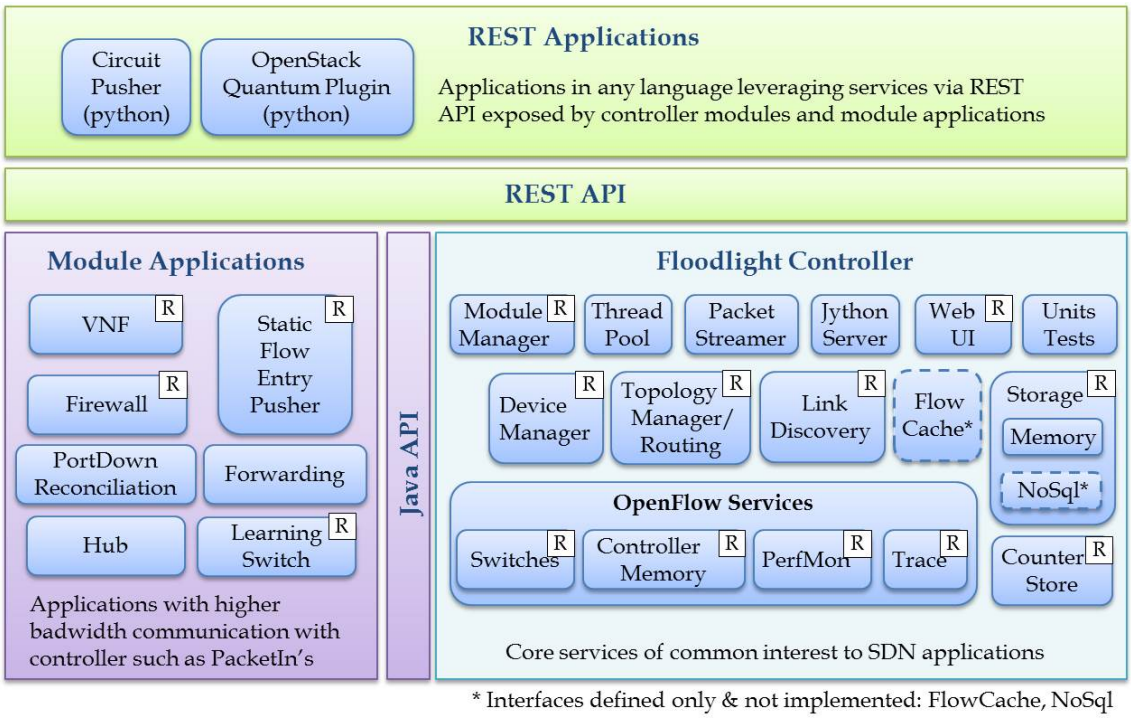

Figura 4.2: Arquitectura de Floodlight y sus aplicaciones.

Se describen brevemente los módulos del controlador y las aplicaciones en los siguientes apartados por ser ésta la herramienta de software utilizada en la etapa de experimentación.

4.3.1.1. Módulos del Controlador

Implementan funciones que son comunes a la mayoría de aplicaciones. Los más importantes son:

- **FloodlightProvider:** Soporta dos funciones importantes. La primera es tratar el manejo de conexiones a los conmutadores y traducir los mensajes OpenFlow en eventos que otros módulos pueden *escuchar*. La segunda función es decidir el orden en que los mensajes OpenFlow (por ejemplo, *Packet-in*, *Flow-Removed*, etc.) son enviados a los módulos que *escuchan* estos mensajes.

- **OFSwitchManager:** Es un módulo diseñado para administrar todos los conmutadores OpenFlow conectados al controlador.

- **DeviceManagerImpl:** Mantiene un registro de los dispositivos cuando éstos se mueven a través de la red y define el dispositivo de destino de los flujos generados en el controlador.

- **LinkDiscoveryManager:** Es el responsable de descubrir y mantener el estado de los enlaces en la red OpenFlow.

- **TopologyService:** Mantiene la información relacionada con la topología de la red en el controlador, así como las funciones de encaminamiento en la red.

- **RestApiServer:** Permite a los módulos presentar una REST API sobre HTTP (*Hypertext Transfer Protocol*).

Los módulos restantes aportan, entre otras funcionalidades, las siguientes: la ejecución periódica de tareas, la gestión del almacenamiento y el manejo de paquetes serializados.

4.3.1.2. Módulos de Aplicaciones

Son módulos en los que se implementan aplicaciones de uso común. Los principales son:

- **VirtualNetworkFilter:** Permite segmentar múltiples redes lógicas de capa 2. Es compatible con OpenStack [Ost], una plataforma de código abierto que sirve para crear una infraestructura de computación en la nube.

- **Forwarding:** Sirve para reenviar paquetes entre dos dispositivos. Los dispositivos origen y destino serán clasificados por el servicio *IDeviceService*.

- **Firewall:** Es un módulo que refuerza las reglas definidas como listas de control de acceso (*Access Control List* o ACL) en los dispositivos OpenFlow, creando nuevas entradas en la tabla de flujo y monitorizando el comportamiento de los paquetes entrantes (*packet-in*).

- **Port Down Reconciliation:** Está implementado para eliminar o corregir entradas en la tabla de flujo referidas a puertos que dejaron de estar activos. Se detectan mediante una actualización (de tipo *PORT_DOWN*) proporcionada por el módulo para el descubrimiento de redes.

- **ACL (*Access Control List*):** Comparado con el módulo de cortafuegos (*Firewall*), la aplicación ACL trabaja de forma proactiva, administrando las entradas en las tablas de flujo en busca de evitar errores de configuración.

4.3.2. OpenDaylight

OpenDaylight [Odl] es una plataforma abierta para la programación de aplicaciones SDN y Virtualización de Funciones de Red (*Network Functions Virtualization* o NFV) para redes de cualquier tamaño o escala. Este controlador SDN tiene una fuerte integración con la plataforma OpenStack. Está escrito en Java.

OpenDaylight es una combinación de componentes que incluyen controlador, interfaces y aplicaciones. En su versión más reciente (Helium), ha evolucionado en otras áreas incluyendo la alta disponibilidad, creación de clústeres y seguridad, así como reforzando y adicionando nuevos protocolos.

Este proyecto ha contado con un amplio respaldo de la industria, enfocándose en crear un marco de trabajo abierto para construir aplicaciones SDN y NFV. No se limita a las innovaciones de OpenFlow, sino que está abierta a otros protocolos.

OpenDaylight cuenta, al igual que Floodlight, con una arquitectura modular definida en [MVTG14]. La arquitectura base se compone de tres capas: Interfaces para los protocolos de tipo *SouthBound* (SB), la capa de adaptación de servicios (*Service Adaption Layer o* SAL), y las interfaces de tipo *NorthBound* (NB) para comunicarse con las aplicaciones. El diseño arquitectónico expuesto ha ido experimentando cambios y mejoras en las sucesivas versiones de OpenDaylight (en particular, en la versión *Helium* [Odl]). No obstante, conserva los principios de modularidad y diseño basado en modelos. En la Figura 4.3 se muestra una vista de la arquitectura que se utiliza en la versión *Helium*.

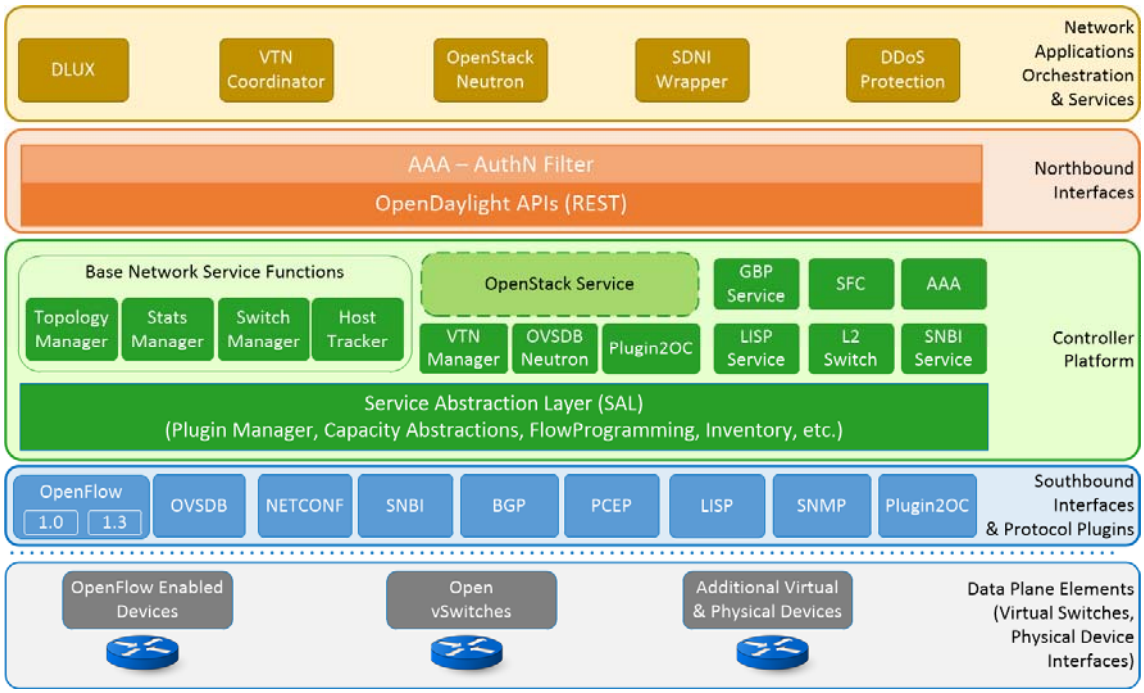

Figura 4.3: Arquitectura de OpenDaylight.

Las capas de la arquitectura definida en *Helium* son:

- **Network Applications & Orchestration Services:** Es la capa superior del modelo y contiene la lógica de gestión de la red que le sirve para controlar y monitorizar su comportamiento. Asimismo, soporta soluciones complejas para entornos en la nube y virtualización de funciones de red (NFV).

- **Controller Platform:** Es la capa intermedia en la que se desarrollan las funciones de abstracción. Proporciona un conjunto de interfaces de programación a la capa de aplicación *(northbound interface*) que vincula a las aplicaciones con las funciones que controlan el comportamiento de los dispositivos de red, mediante una interfaz (*southbound interface*) que implemente algún protocolo para ese fin (por ejemplo, OpenFlow).

- **Physical & Virtual Network Devices:** Es la capa inferior del modelo y se compone de los dispositivos de red físicos y virtuales que implementan la red mediante conexiones entre nodos.

OpenDaylight soporta actualmente las versiones 1.0 y 1.3 del protocolo OpenFlow.

4.3.3. NOX/POX

El controlador NOX [Nx1] fue creado en 2008 y ha sido uno de los primeros controladores SDN de código abierto. Ha contado con contribuciones de la Universidad de Stanford, UC Berkeley e ICSI. NOX proporciona una API en C++ para OpenFlow (versión 1.0) y un modelo de programación asíncrono basado en eventos [NG13].

NOX es un controlador y un marco de trabajo para crear aplicaciones para SDN. Aunque sólo proporciona módulos para OpenFlow, puede extenderse a otros protocolos. Ha sido usado principalmente en entornos académicos y de investigación.

Los criterios de diseño e implementación de NOX se presentan en [GKP+08]:

- **Componentes:** Los componentes de la arquitectura NOX son los conmutadores OpenFlow, un servidor que ejecuta NOX (controlador) y una base de datos que almacena la vista global de la red.

- **Granularidad:** La arquitectura de NOX es granular y permite compensar aspectos de escalabilidad y flexibilidad.

- **Abstracción de conmutadores:** Se ha adoptado el modelo definido en el protocolo OpenFlow, representando el comportamiento de los conmutadores en tablas de flujo.

- **Operación:** Los paquetes que coincidan con entradas de las tablas de flujo actualizarán los contadores y realizarán las acciones establecidas. Los que no, serán enviados al controlador.

- **Escalabilidad:** Considera un manejo eficiente de escalas de tiempo y requerimientos de consistencia de datos para asegurar la escalabilidad.

POX es la implementación de NOX en Python que aporta las siguientes ventajas:

- Cuenta con una interfaz OpenFlow para Python.

- POX tiene componentes reusables para la selección de rutas, descubrimiento de topologías, etc.

- Soporta las mismas interfaces gráficas de usuario (GUI) y herramientas de visualización que NOX.

- Las aplicaciones presentan un buen rendimiento comparadas con aquellas escritas en NOX.

4.3.4. Trema

Trema [Treb] es un controlador de red de código abierto que fue originalmente desarrollado por NEC [Nc1] y posteriormente se ha ido desarrollando con contribuciones de la comunidad.

A diferencia de los controladores OpenFlow convencionales, el controlador Trema proporciona servicios básicos de infraestructura de red como parte de su núcleo de desarrollo (*core modules*), que soportan, a su vez, el desarrollo de módulos creados por el usuario (*Trema apps*) [Trea]. Los módulos pueden implementarse en Ruby o C. Éste último es recomendado cuando el rendimiento es una prioridad [NG13].

El controlador está programado en base a eventos. Asimismo, cuenta con un sistema de intercambio de mensajes (*Inter-Process Communication* o IPC) para comunicar a los módulos del usuario y del sistema, como se puede ver en la Figura 4.4. Otros módulos esenciales son las bibliotecas de registro de sucesos (*logging)*, los analizadores de paquetes, las bibliotecas de gestión de tablas *hash* y listas enlazadas, entre otros.

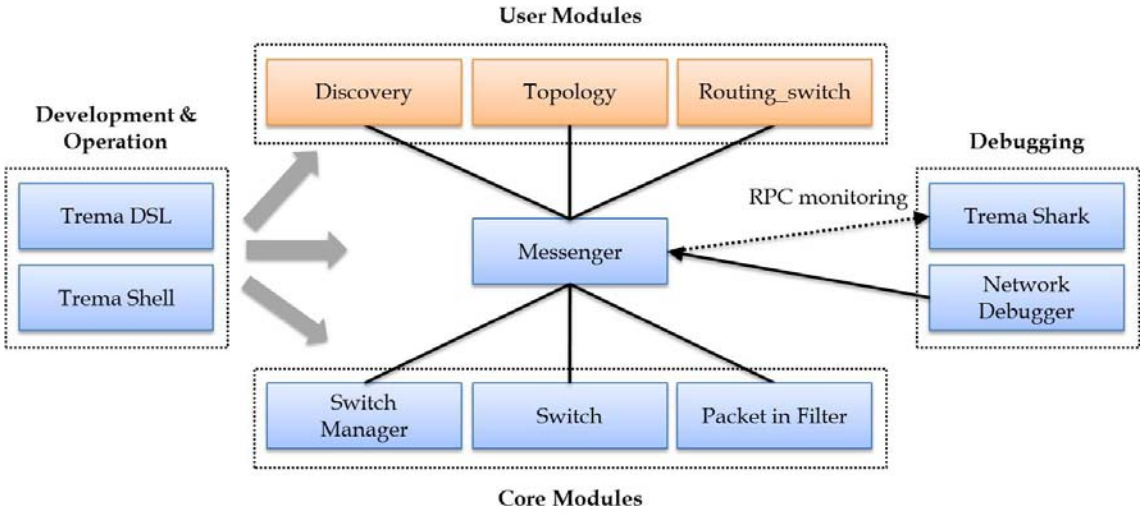

Figura 4.4: Relaciones entre módulos de usuario y de sistema en Trema.

4.3.5. Ryu

Ryu [Ry1] es un entorno de trabajo basado en componentes para el desarrollo de aplicaciones SDN. Ryu proporciona componentes software y una API que, en conjunto, facilitan el desarrollo de aplicaciones para la gestión de una red SDN. Ryu soporta varios protocolos para el manejo de dispositivos de red, tales

como OpenFlow (versiones 1.0 a 1.4), Netconf, OF-config, entre otros. Está, a su vez, fuertemente integrado con OpenStack. Está escrito íntegramente en Python, lo que puede constituir un factor a considerar si lo que se requiere es desarrollar aplicaciones de alto rendimiento que suelen ser mejor implementadas en lenguajes compilados. Ryu implementa también una interfaz REST para interactuar con las aplicaciones.

4.3.6. Mininet

Aunque no se trata estrictamente de un controlador SDN, Mininet [Mi1] es una plataforma que permite emular una red OpenFlow completa en un mismo computador. Mininet usa un modelo de virtualización basado en procesos ligeros y espacios de nombres para ejecutar muchos *hosts* y conmutadores OpenFlow sobre un mismo *kernel* de sistema operativo [Azo13]. Mininet conecta conmutadores y *hosts* utilizando pares de interfaces virtuales Ethernet (*veth*). Esta herramienta software reduce considerablemente el tiempo de desarrollo, depuración, prueba y despliegue de las aplicaciones SDN. Es por ello que será utilizada en la fase de experimentación de este trabajo.

En la Figura 4.5 se muestra un ejemplo de red de dos *hosts* y un controlador implementada sobre Mininet [LHM10]. Se puede apreciar en la figura la presencia de los componentes y conexiones basadas en pares de interfaces virtuales Ethernet para una red de dos *hosts*.

Las características de Mininet son [LHM10] [Azo13]:

- **Enlaces:** Un par de interfaces virtuales Ethernet (*veth*) actúa como un cable conectando dos interfaces. Estos pares pueden ser conectados a conmutadores virtuales (por ejemplo, *Linux Bridge*) o a conmutadores OpenFlow.

- **Hosts:** Los espacios de nombres de red son los contenedores del estado de la red. Ellos proporcionan procesos (y grupos de procesos) propietarios de las interfaces, puertos y tablas de encaminamiento. Por ejemplo, dos servidores web pueden coexistir en un único sistema, ambos aceptando conexiones en sus respectivas interfaces *eth0* y en el puerto 80. Un *host* en Mininet es una consola (por ejemplo, *bash*) situada en su propio espacio de nombres de red mediante una llamada al sistema.

- **Conmutadores:** Soportan el protocolo OpenFlow con las mismas disciplinas de procesamiento de paquetes que los conmutadores hardware.

- **Controladores:** Los controladores pueden estar en la red real o en la red emulada, siempre que el equipo que ejecuta los conmutadores tenga conectividad IP con el controlador.

- **API:** Aunque es posible usarla desde la línea de comandos para la creación de entornos de red con una amplia variedad de opciones de configuración, Mininet brinda una API para Python que permite la creación de redes y amplía sus capacidades de experimentación.

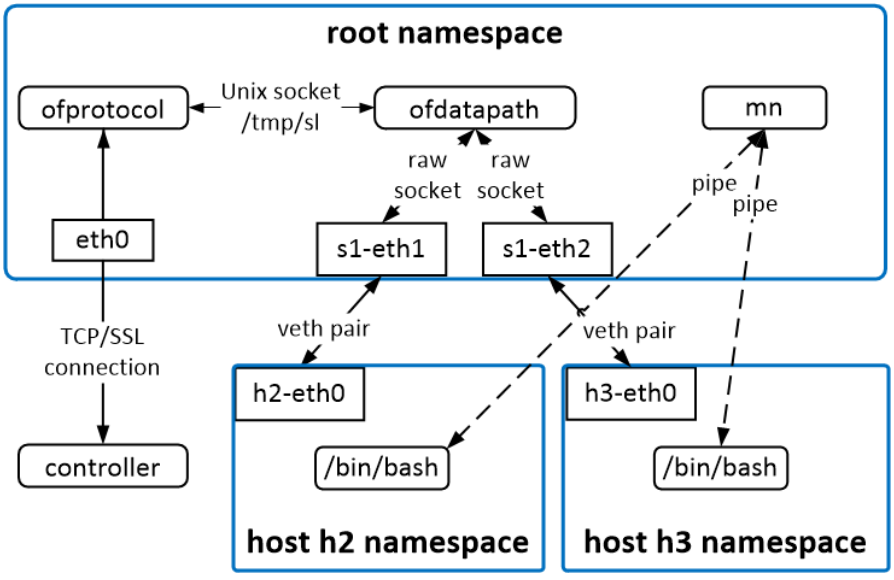

Figura 4.5: Red OpenFlow creada en Mininet.

4.4. Comparativa de Controladores.

En [GCRVACCB14], [KZMB14] y [Pri15b] se examinan algunas características en los controladores SDN más comunes con el fin de establecer una comparativa desde un enfoque global. La Tabla 4.1 muestra una síntesis de las mismas.

Característica	NOX/ POX	Ryu	Trema	Floodlight	OpenDaylight
Interfaces	SB (OpenFlow)	SB (OpenFlow) + OVSDB	SB (OpenFlow)	SB (OpenFlow), NB (Java y REST)	SB (OpenFlow y otros), NB (REST y Java RPC)
Virtualización	Mininet y Open vSwitch	Mininet y Open vSwitch	Herramienta propia de emulación	Mininet y Open vSwitch	Mininet y Open vSwitch
Soporte OpenFlow	v 1.0	v 1.0, 1.2, 1.3 y 1.4	v 1.0	v 1.0 y 1.3	v 1.0 y 1.3
GUI	Sí	Sí	Sí	Sí	Sí
REST API	No	Sí (SB)	No	Sí	Sí
Código Abierto	Sí	Sí	Sí	Sí	Sí
Documentación	Pobre	Media	Media	Buena	Media
Lenguaje soportado	Python	Python	C/Ruby	Java + Cualquier lenguaje que soporte REST	Java
Plataformas soportadas	Linux, Mac OS y Windows	Linux	Linux	Linux, Mac OS y Windows	Linux
Soporte TLS	Sí	Sí	Sí	Sí	Sí
Soporte para OpenStack	No	Fuerte	Débil	Media	Media
Multiprocesos	Sí	No	Sí	Sí	Sí
Distribuido	No	Sí	No	Sí	Sí

Tabla 4.1: Comparación de controladores SDN.

Según los requerimientos particulares de cada aplicación debe hacerse una selección cuidadosa de los mejores controladores y entornos de desarrollo, que satisfagan las necesidades de gestión de una red desde distintas perspectivas.

5. COMUNICACIÓN MULTIMEDIA EN TIEMPO REAL

5.1. Introducción

Internet no se creó teniendo en cuenta el transporte de tráfico en tiempo real. Sin embargo, las necesidades de comunicación motivaron que se desarrollen soluciones multimedia con el objetivo de transmitir datos en tiempo real. Algunos problemas que debieron ser afrontados fueron: las demandas de ancho de banda, la transmisión a uno o múltiples destinatarios a la vez, mecanismos para la sincronización de datos y para asegurar el orden en la entrega de paquetes, la gestión de *buffers* o memorias temporales, entre otros.

En respuesta a esas necesidades se crearon distintos protocolos para la transmisión de datos multimedia en tiempo real sobre redes IP. Este capítulo se ocupa de los protocolos RTSP, RTP Y RTCP, por ser los de mayor relevancia en el desarrollo de los experimentos.

5.2. Protocolos Multimedia

5.2.1. RTSP

El protocolo RTSP (*Real Time Streaming Protocol*) fue definido en la RFC 2326 [SRL98]. Es un protocolo de nivel de aplicación para el control de flujos de datos en tiempo real, no orientado a conexión, que se utiliza para definir cómo se hará el envío de datos entre el cliente y el servidor. RTSP controla que la entrega de datos se realice correctamente, dado que el tipo de contenido con el que se trabaja normalmente al hacer *streaming* es sensible a la sincronización. Usualmente, se compara RTSP con una especie de mando a distancia de red para controlar las operaciones de un servidor multimedia.

RTSP hace uso de sesiones etiquetadas por un identificador único. El protocolo establece y controla uno o más flujos sincronizados de datos (como audio y vídeo), intentando conseguir siempre que el flujo de datos sobre la red IP sea lo más eficiente posible. RTSP es independiente del protocolo de transporte y puede funcionar tanto sobre TCP o UDP.

Los flujos de datos administrados por RTSP suelen usar RTP, pero el modo de operación de RTSP es independiente del mecanismo de transporte usado para transmitir los datos. Sin embargo, en la mayoría de casos las operaciones de control se hacen sobre TCP y la transmisión de datos con RTP sobre UDP.

RTSP soporta un conjunto de operaciones para controlar el flujo de datos. Las principales son [SRL98] [Zur04]:

- Obtención de contenidos multimedia desde un servidor: El cliente puede solicitar una descripción de la presentación vía HTTP o cualquier otro método. Seguidamente, puede establecer una sesión para la transmisión.

- Invitación de un servidor multimedia a una conferencia: Un servidor de *streaming* puede ser invitado a unirse a una conferencia ya existente.

- Adición de contenido multimedia a una presentación existente: Particularmente para presentaciones en directo, es muy útil si el servidor puede notificar al cliente sobre los nuevos contenidos adicionales disponibles.

RTSP soporta también un conjunto de peticiones durante la transmisión del contenido. Las principales son:

- OPTIONS: Esta petición puede enviarse en cualquier momento y no influye en el estado de la sesión. Es utilizada en el inicio de la sesión con los parámetros necesarios, como el número que se asigna y cierta información del servidor como la versión o los métodos soportados.

- DESCRIBE: El cliente obtiene una descripción de la presentación u objeto multimedia a transmitir.
- SETUP: El cliente solicita al servidor la asignación de recursos para el inicio de una sesión.
- PLAY: El cliente solicita el inicio de datos sobre una sesión previamente establecida.
- PAUSE: El cliente solicita una pausa en la transmisión sin que se liberen recursos.
- TEARDOWN: El cliente solicita el final de la transmisión y la liberación de los recursos asignados.

En la Figura 5.1 se esquematiza el intercambio de mensajes entre un cliente y un servidor a través de una sesión RTSP.

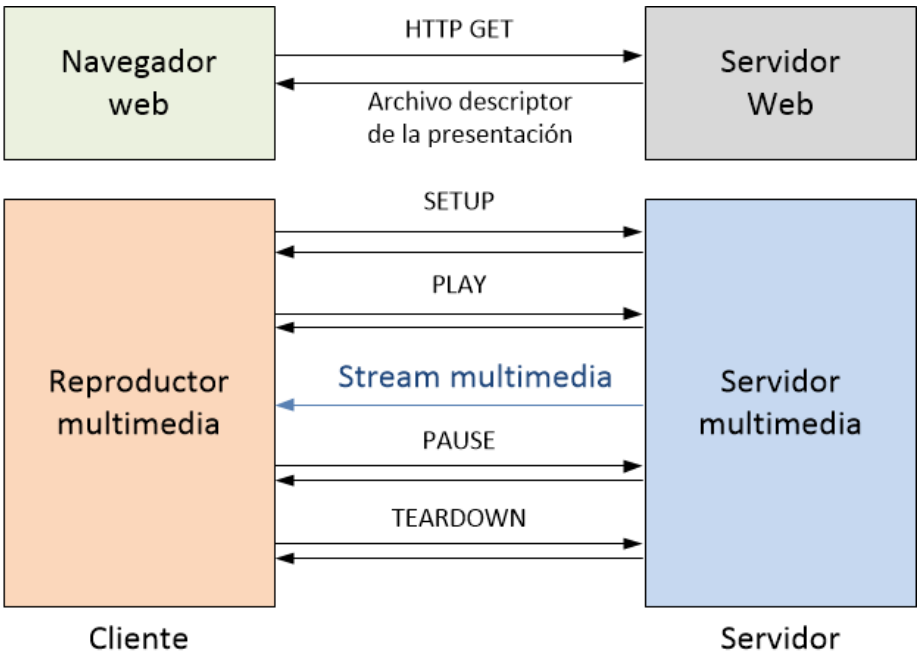

Figura 5.1: Comunicación RTSP entre un cliente y un servidor.

5.2.2. RTP

El protocolo de Transporte en Tiempo Real (*Real-time Transport Protocol* o RTP) [SCFJ03] está definido en la RFC 3550 y se diseñó para la transferencia de datos secuenciales en tiempo real. RTP fue en un principio creado para emisiones *multicast* (aunque también se puede utilizar en emisiones *unicast*) y para transmitir vídeo bajo demanda, así como servicios interactivos tales como telefonía en Internet. En la capa de transporte puede utilizar cualquier protocolo, aunque a menudo se utiliza sobre UDP.

RTP trabaja conjuntamente con el protocolo RTCP que le otorga retroalimentación sobre la calidad de los datos transmitidos.

5.2.2.1. Encabezado RTP

Los datos en una sesión RTP se transmiten como una serie de paquetes IP que se originan en un nodo emisor. Cada paquete de datos RTP tiene dos partes: un encabezado y la carga útil de datos (*payload*). En la Figura 5.2 se muestra el encabezado de un paquete RTP.

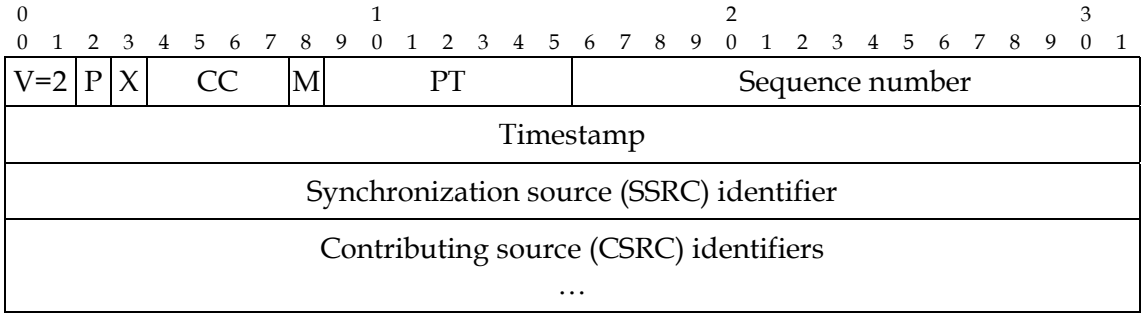

Figura 5.2: Encabezado de un paquete RTP.

Los campos del encabezado son:

- **Versión (V),** 2 bits: La versión del protocolo.

- **Relleno (P - Padding),** 1 bit: Si el bit del relleno está activado, hay uno o más bytes al final del paquete que no son parte de la carga útil.

- **Extensión (X - Extension),** 1 bit: Si el bit de extensión está activado, entonces el encabezado fijo es seguido por una extensión del encabezado.

- **Recuento de CSRC (CC),** 4 bits: El número de identificadores CSRC que sigue el encabezado fijo. Si la cuenta CSRC es cero, entonces la fuente de sincronización es la fuente de la carga útil.

- **Marcador (M - Marker),** 1 bit: La interpretación del marcador está definida por un perfil adicional que permite la identificación de eventos significativos en el flujo de paquetes.

- **Tipo de carga útil (PT – Payload Type),** 7 bits: Determina el tipo de la carga útil de datos y sirve para que la aplicación sepa cómo procesarlos.

- **Número de Secuencia,** 16 bits: Es un número que se incrementa en uno por cada paquete enviado. Se utiliza por el receptor para detectar pérdida de paquetes y para restablecer la secuencia de los mismos. El número de secuencia inicial es aleatorio.

- **Marca de tiempo (Timestamp),** 32 bits: Refleja el instante de muestreo del primer byte en la carga útil. Varios paquetes consecutivos pueden tener la misma marca de tiempo si son generados a nivel lógico en el mismo instante. Por ejemplo, si son todos parte de la misma trama de vídeo.

- **SSRC,** 32 bits: Identifica la fuente de sincronización.

- **CSRC,** 32 bits: Identifica las fuentes para generar la carga útil. El número de fuentes está indicado en el campo CC.

5.2.2.2. Funcionamiento del Protocolo RTP

Los paquetes transmitidos por Internet sufren un retardo y un *jitter* impredecibles, afectando al rendimiento de las aplicaciones en tiempo real. Este es el motivo por el que RTP proporciona un mecanismo para establecer marcas de tiempo (*TimeStamping*) que controlen esta variación. Las marcas de tiempo son la información más importante de los paquetes en tiempo real. El emisor establece las marcas de tiempo y el receptor las utiliza para reconstruir la secuencia de tiempo original.

Las marcas de tiempo se utilizan también para sincronizar distintos flujos como información de audio y vídeo en MPEG. Sin embargo, RTP por sí sólo no es responsable de la sincronización, ya que esta función está destinada a la aplicación. Como UDP no garantiza la entrega de paquetes en orden correcto, se utilizan los números de secuencia para ordenar los paquetes y detectar pérdidas. Los números de secuencia son de ayuda también para ordenar paquetes que contienen la misma marca de tiempo.

Las aplicaciones RTP están a menudo divididas en las que necesitan poder recibir datos de la red (clientes RTP) y las que necesitan poder transmitir datos a través de la red (servidores RTP). Algunas aplicaciones hacen ambas funciones; por ejemplo, las aplicaciones de teleconferencia envían y reciben datos al mismo tiempo. Mientras RTP no facilita mecanismos para asegurar entrega oportuna u otra calidad de garantía de servicio, estas funciones son soportadas por un protocolo de control (RTCP), el que se explica a continuación.

5.2.3. RTCP

El Protocolo de Control RTP (*RTP Control Protocol* o RTCP) [SCFJ03] está definido, junto con RTP, en la RFC 3550. Es un protocolo de control diseñado

para operar junto con RTP. Se basa en la transmisión periódica de paquetes de control por parte de todos los participantes de una sesión. Las sesiones en RTP y RTCP se establecen utilizando puertos de red separados. Usualmente, RTP utiliza un puerto par y RTCP usa el siguiente número de puerto impar disponible.

5.2.3.1. Encabezado RTCP

RTCP define cinco tipos de paquetes para el transporte de información de control:

- **RR** (*Receiver Report*): Los RR son generados por aquellos participantes que no son emisores activos. Indica el número de paquetes recibidos, el número de paquetes perdidos, la fracción de paquetes perdidos, el *jitter*, entre otros parámetros de calidad.

- **SR** (*Sender Report*): Los SR son generados por emisores activos. Además de mantener la calidad de la recepción como en RR, contienen una sección de información del emisor, proporcionando información de sincronización, contadores de paquetes acumulados y número de paquetes enviados.

- **SDES** (*Source Description Items*): Contienen información para describir las fuentes.

- **BYE**: Indica el final de la participación.

- **APP** (*Application Specific Functions*): Funciones específicas de aplicación.

En la Figura 5.3 se muestra el encabezado de un paquete RTCP de tipo RR (*Receiver Report*) [Koi00].

V	P	RC	PT=201	Length
SSRC of the sender				
SSRC of the first source				
Fract. lost		Cum. No of packets lost		
Ext. Highest sequence number received				
Interarrival jitter estimate				
Last sender report timestamp (LSR)				
Delay since last sender report (DLSR)				
…				
Last reception report block				

Figura 5.3: Encabezado de un paquete RTCP tipo RR.

Se explican los campos más relevantes del informe a efectos de la presente investigación [Rib07]:

- **Interarrival Jitter:** El *jitter* está definido como la variación del retardo que se produce en los tiempos entre llegadas de los paquetes RTP. Se mide en unidades de marcas de tiempo (*timestamp units*). Estas variaciones pueden ser causadas por factores de congestión en la red o las aplicaciones.

- **Fraction Lost**: La fracción de paquetes RTP procedentes de un origen (SSRC_n) perdidos desde que se recibió el último SR o RR. Se obtiene de dividir el número de paquetes perdidos entre el número de paquetes esperados. Si la fracción es negativa, ésta se establece automáticamente a cero.

5.2.3.2. Funciones del Protocolo RTCP

RTCP desempeña cuatro funciones principales:

- Informar sobre la calidad en la entrega de datos y sobre los participantes de la sesión. Esta información se envía a través de informes de emisor y receptor.

- Usar identificadores denominados nombres canónicos (CNAME) por cada origen de datos RTP, que le permiten llevar un registro de cada participante.

- Establecer la frecuencia de envío de los paquetes RTCP según el número de participantes encontrados en una sesión.

- Transmitir la mínima información de control necesaria para optimizar el tráfico entre los participantes.

Los servicios que ofrece RTCP en una comunicación multimedia son: monitorización de la calidad del servicio y control de congestión, identificación de la fuente, sincronización y escalabilidad en la información de control. El protocolo RTP limita el tráfico de control al 5% de todo el tráfico de la sesión.

5.3. Medición de la Calidad de un Vídeo

La medición de la calidad de un vídeo debe estar basada en la calidad percibida por los usuarios. Esa percepción es la que finalmente resulta determinante. Esta impresión intuitiva de un usuario que ve un vídeo es obtenida mediante métricas de calidad subjetivas. Aunque este tipo de métricas proporciona información valiosa, su evaluación es costosa dado que se requiere contar con personas que emitan su opinión sobre la calidad de un vídeo. Asimismo, se requiere del equipamiento adecuado. Frente a estas limitaciones se crearon métricas objetivas que emulen la percepción de la calidad de un sistema visual humano (*Human Visual System* o HVS) [GKKW04]. En la presente investigación se ha recurrido a dos métricas: PSNR y MOS.

5.3.1. PSNR

La métrica objetiva más usada para medir la calidad del vídeo es el PSNR (*Peak Signal-to-Noise Ratio*), que se aplica a cada imagen incluida en el vídeo [KRW03]

[GKKW04]. Esta métrica se deriva del SNR (*Signal-to-Noise Ratio*). PSNR compara la máxima energía posible de la señal con respecto a la energía de ruido presente en cada fotograma. El PSNR calculado entre una imagen origen (*s*) y una imagen destino (*d*) se obtiene aplicando la Ecuación 5.1 [GKKW04]:

$$PSNR(s,d) = 20\log\frac{V_{peak}}{MSE(s,d)}[db] \qquad (5.1)$$

donde V se calcula según la Ecuación 5.2:

$$V_{peak} = 2^k - 1 \qquad (5.2)$$

siendo k la profundidad del color en bits,

y MSE se define como el error cuadrático medio de *s* y *d*.

5.3.2. MOS

MOS (*Mean Opinion Score*) mide la calidad de vídeo percibida por el ojo humano. Es un método de medición subjetivo mediante el cual se recopila la impresión de las personas (juicio de expertos) que ven un determinado vídeo y otorgan una valoración cualitativa entre 1 (malo) y 5 (excelente) [KRW03]. En la Tabla 5.1 se muestra la escala de valores de este indicador.

Escala	Calidad	Degradación
1	Excelente	Imperceptible
2	Buena	Perceptible, pero no incómoda
3	Aceptable	Ligeramente incómoda
4	Pobre	Incómoda
5	Mala	Muy incómoda

Tabla 5.1: Escala de valores del MOS [KRW03].

Otra posibilidad para calcular el MOS es obteniéndolo a partir del valor del PSNR mediante una escala de correspondencia entre los intervalos de valores del PSNR con los del MOS. Esa equivalencia se muestra en la Tabla 5.2.

PSNR [dB]	MOS
> 37	5 (Excelente)
31 - 37	4 (Buena)
25 - 31	3 (Aceptable)
20-25	2 (Pobre)
< 20	1 (Mala)

Tabla 5.2: Equivalencia entre el MOS y PSNR [KRW03].

6. ARQUITECTURA PARA LA MEJORA DE LA CALIDAD MULTIMEDIA

6.1. Introducción

El paradigma SDN desacopla los planos de datos y de control en los dispositivos de red y establece una visión centralizada de la red. En este capítulo se plantea extender la arquitectura SDN con el fin de mejorar la calidad de experiencia de un usuario en la visualización de vídeo transmitido desde un servidor multimedia. Se ha definido esta arquitectura de forma genérica para luego implementarla en un escenario que será usado en la experimentación. Aquello involucra la selección específica de protocolos de comunicación multimedia, interfaces de programación, sistema operativo de red, entorno de programación, entre otros, que darán lugar al escenario de prueba.

6.2. Descripción de la Arquitectura

SDN proporciona un esquema compuesto por tres capas: infraestructura, control y aplicación. En esta investigación se extiende la arquitectura SDN introduciendo componentes funcionales que se muestran en la Figura 6.1. En la capa de aplicación se introduce un Servicio de Gestión de Recursos, implementado como un componente que es capaz de comunicarse con el controlador SDN a través de alguna interfaz de comunicación. En este modelo se optó por REST API. En la capa de control se implementan dos módulos, uno para la monitorización de la red y otro para el cálculo de rutas óptimas que hagan más eficiente la transmisión del contenido multimedia. Finalmente, en los nodos origen y destino de la comunicación se implementan componentes que interactúen con el software multimedia y sean capaces de soportar el protocolo PCMS, definido para el intercambio de mensajes entre los Agentes de Gestión de Recursos y el Servicio de Gestión de Recursos.

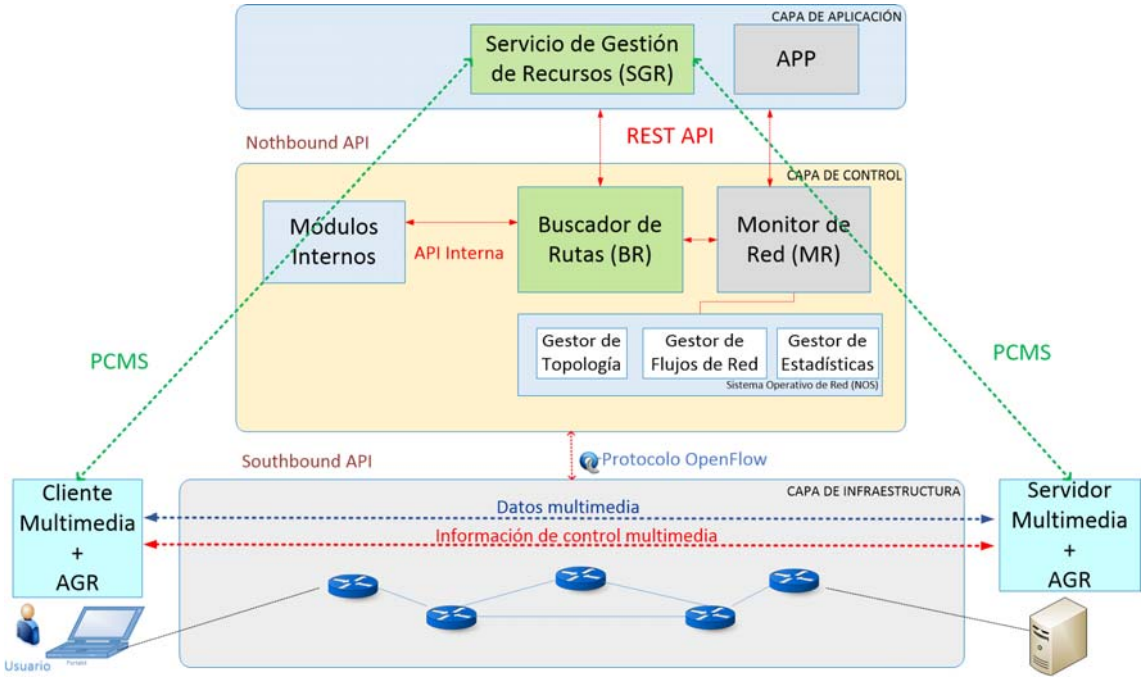

Figura 6.1: Arquitectura para la mejora de la calidad multimedia en redes SDN.

Los elementos presentes en cada capa y las funciones que cada uno desempeña se explican a continuación.

6.2.1. Capa de Infraestructura

La capa de infraestructura incluye no sólo los dispositivos de red, sino también los dispositivos terminales (móvil, ordenador portátil). La comunicación entre la capa de infraestructura y la capa de control se hace a través de la API proporcionada por el protocolo OpenFlow (*southbound* API).

6.2.2. Capa de Control

En la capa de control el sistema operativo de red [FRZ13] procesa la información recibida por los mensajes OpenFlow y proporciona una abstracción de alto nivel de la infraestructura de red. El Gestor de Topología identifica los dispositivos conectados en la red y los enlaces existentes entre ellos. La topología se representa como un grafo G (N, A), donde N representa el conjunto

de conmutadores y A el conjunto de enlaces entre dispositivos. El Gestor de Flujos de Red utiliza mensajes OpenFlow para configurar las tablas del flujo en los conmutadores. El Gestor de Estadísticas organiza la información proporcionada por los contadores internos de los conmutadores. Por su parte, el Monitor de Red utiliza módulos del NOS y calcula métricas que reflejen el estado actual de la red (congestión, pérdida de paquetes). En [VADK14] y [TGG10] se describen esquemas de monitorización similares basados en recopilación de estadísticas. Del mismo modo, el componente Buscador de Rutas es el responsable de evaluar y definir rutas entre los dispositivos origen y destino, utilizando la información proporcionada por el Monitor de Red y por el Servicio de Gestión de Recursos. El Buscador de Rutas recibe información desde el Monitor de Red a través de una API interna (propia del NOS) y desde el Servicio de Gestión de Recursos a través de una REST API (*northbound* API).

6.2.3. Capa de Aplicación

En la capa de aplicación el Servicio de Gestión de Recursos (SGR) intercambia mensajes con los Agentes de Gestión de Recursos (AGR), presentes tanto en el cliente como en el servidor, a través de un protocolo propuesto, al que se ha denominado PCMS (Protocolo de Control Multimedia en SDN). El agente del lado del cliente recopila información de control que le ofrece el protocolo multimedia (por ejemplo, RTCP) y envía dicha información al SGR, que a su vez la envía al Buscador de Rutas para el cálculo de la mejor ruta. El cliente y el servidor multimedia efectúan la transmisión sobre la red SDN según el protocolo seleccionado por la aplicación.

6.3. El Protocolo PCMS

Para llevar a cabo el intercambio de mensajes entre el SGR y el AGR, se propone el protocolo PCMS, según la estructura que se muestra en la Figura 6.2.

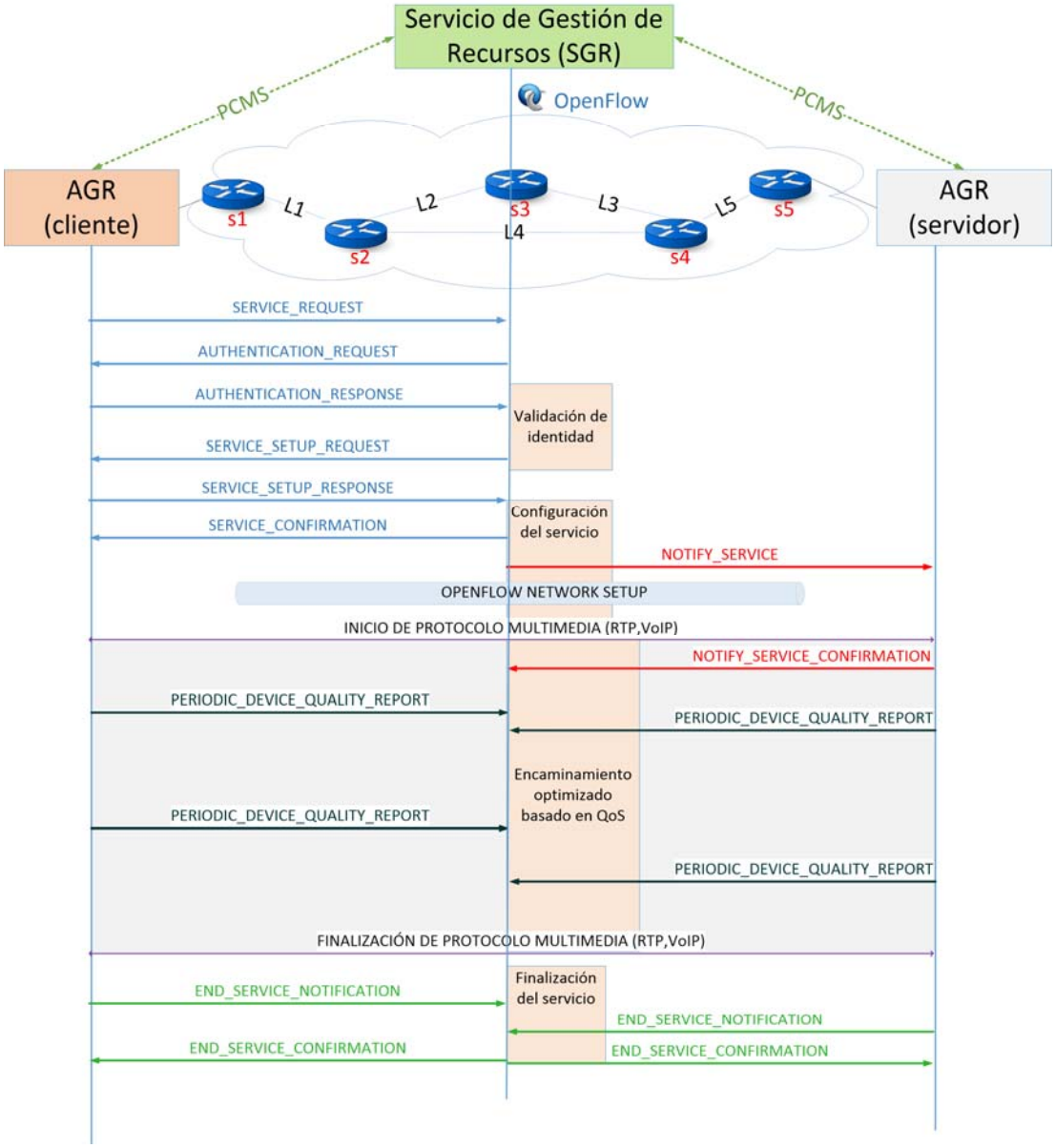

Figura 6.2: Descripción del protocolo PCMS.

En el protocolo se distinguen tres momentos diferenciados en la transmisión de datos multimedia (antes, durante y después), cuyos procesos de comunicación se detallan a continuación.

6.3.1. Antes de la Transmisión

El AGR cliente inicia la conexión hacia el SGR para solicitar un recurso multimedia mediante un mensaje SERVICE_REQUEST. El Servicio de Gestión

de Recursos, en respuesta, envía una petición de autenticación al cliente a través de un mensaje de tipo AUTHENTICATION_REQUEST. El cliente responde con un mensaje AUTHENTICATION_RESPONSE. El protocolo queda abierto a cualquier esquema de autenticación, con la única condición de incorporar las credenciales de autenticación en el mensaje. En la presente propuesta se ha trabajado un esquema simplificado de usuario y contraseña.

El SGR verifica la identificación del cliente y, si es satisfactoria, le solicita los parámetros multimedia del contenido a transmitir a través de un mensaje SERVICE_SETUP_REQUEST. El cliente responde al servidor con un mensaje SERVICE_SETUP_RESPONSE con los correspondientes parámetros multimedia, que incluyen, comúnmente, el nombre del recurso, la tasa de transferencia (*bitrate*), los fotogramas por segundo (*framerate*), la duración u otros parámetros descriptivos.

En esta fase el SGR utiliza el grafo G(N, A) proporcionado por el Gestor de Topología para localizar la posición del servidor multimedia en la red SDN. Una vez localizado, el SGR confirma al cliente el permiso para iniciar la transmisión multimedia mediante un SERVICE_CONFIRMATION. A su vez, notifica al servidor multimedia el inicio de la comunicación mediante un mensaje NOTIFY_SERVICE con los parámetros recibidos del cliente. El AGR del lado del servidor notifica a las capas superiores el inicio del correspondiente protocolo multimedia (*streaming*, VoIP, RTP).

6.3.2. Durante de la Transmisión

Mientras la transmisión está activa, el AGR envía periódicamente informes sobre la calidad del servicio al SGR en mensajes PERIODIC_DEVICE_QUALITY_REPORT. Los parámetros específicos del informe dependen del tipo de servicio y del protocolo multimedia involucrado en la tarea de control. A su vez, el SGR envía esta información periódica al

Buscador de Rutas para que calcule y configure la mejor ruta (*Integrated Optimized Routing*) en los conmutadores OpenFlow. Esta configuración asegura que los recursos de red asignados sean adecuados para mantener la calidad de la transmisión multimedia.

6.3.3. Después de la Transmisión

El fin de la transmisión es detectado por el AGR, que se encarga de notificarlo al SGR para que libere los recursos de red, a través de mensajes (END_SERVICE NOTIFICATION y END_SERVICE_CONFIRMATION).

6.4. Componentes de la Arquitectura y su Implementación

La arquitectura propuesta distingue los componentes en dos categorías: los componentes multimedia y los componentes del controlador de red.

6.4.1. Componentes Multimedia

En esta categoría se plantea la existencia de tres componentes multimedia: software multimedia, AGR y SGR, como puede verse en la Figura 6.1. En cada extremo de la conexión se cuenta con una instancia del software multimedia y el AGR (cliente o servidor, según corresponda). En la Figura 6.3 se muestra la interacción de estos componentes con el SGR, mediante mensajes propios del protocolo PCMS.

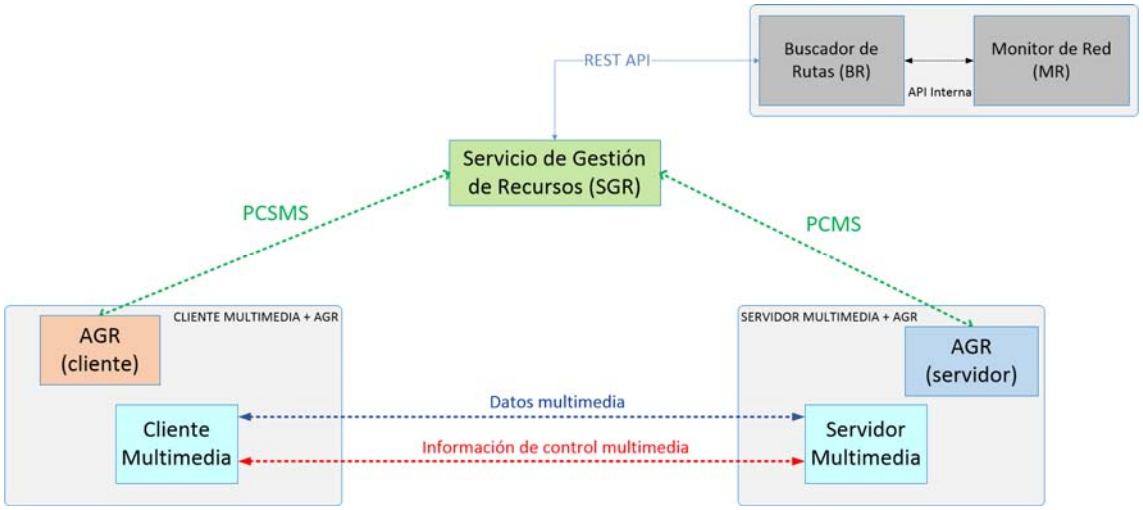

Figura 6.3: Componentes multimedia de la arquitectura.

6.4.1.1. Software Multimedia

La comunicación multimedia puede ser gestionada por cualquier programa que haga uso de protocolos que diferencien los canales de datos y de control (como RTP). La distinción de funciones es importante para que el AGR pueda distinguir el origen de los datos de control que determinen la calidad de la transmisión. Distintas bibliotecas de software pueden ser usadas para este propósito. No es un requisito que se usen puertos distintos para los canales, siempre que el AGR pueda diferenciar el tráfico de datos del tráfico de control.

En la implementación de la arquitectura se ha hecho uso de la herramienta Video Tester 1.0 [NOALS12] para procesar la información multimedia. Video Tester es un *framework* escrito en Python para evaluar la calidad de vídeos trasmitidos sobre una red física o virtual. El esquema de operación de Video Tester se muestra en la Figura 6.4.

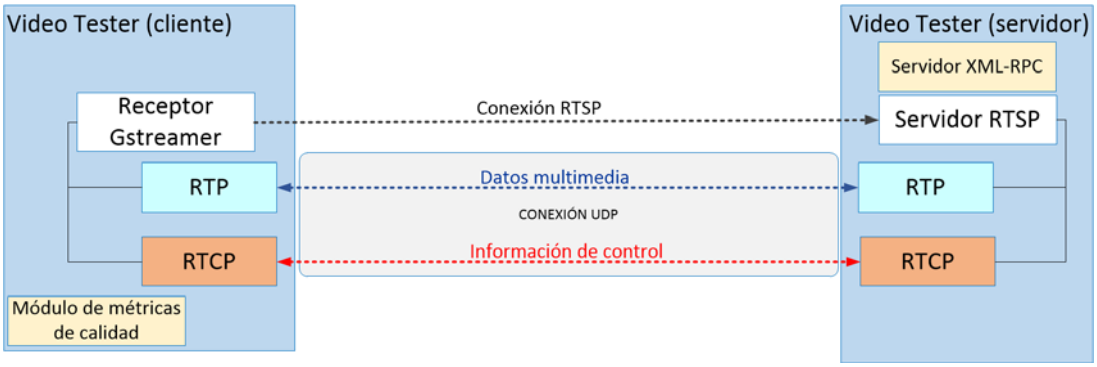

Figura 6.4: Esquema de operación en Video Tester.

El vídeo se envía al cliente a través de una sesión RTSP creada desde Video Tester haciendo uso de las bibliotecas de Gstreamer v 0.10 [Gst]. La sesión RTSP genera el inicio de una sesión RTP sobre UDP (canal de datos multimedia de la Figura 6.4), que, como se ha explicado en el capítulo anterior, trabaja conjuntamente con el protocolo RTCP, el cual se encarga de intercambiar los informes de control entre los participantes (canal de información de control multimedia de la Figura 6.4). En los experimentos se usan los puertos cliente 5532 y 5533 para RTP y RTCP, respectivamente. Estos puertos son asignados aleatoriamente en el servidor en el proceso de negociación RTSP.

El análisis del vídeo se lleva a cabo en Video Tester al culminar la transmisión, mediante el cálculo de métricas de calidad que se determinan en los tres niveles que intervienen en el procesamiento y transmisión de vídeo: nivel de paquetes, nivel de flujo de bits y nivel de imágenes.

6.4.1.2. Agente de Gestión de Recursos (AGR)

La información de control generada por el software multimedia es detectada por el AGR, que identifica y extrae los parámetros relevantes para enviar al controlador SDN. El controlador los usará para introducir mejoras en el comportamiento de la red, que generen un incremento en la calidad de la transmisión. Los parámetros extraídos en el AGR son enviados al SGR en

mensajes del tipo PERIODIC_DEVICE_QUALITY_REPORT, como se explicó en la sección 6.2.2. La importancia de los AGR en ambos extremos de la conexión radica en gestionar los mensajes del protocolo PCMS.

El AGR cliente se ha implementado en un script, escrito en Python (v. 2.7.4), que gestiona el intercambio de mensajes del protocolo PCMS con el SGR. A su vez, el AGR se encarga de lanzar la aplicación cliente (Video Tester) y ejecutar en paralelo el análisis de tráfico, que consiste en filtrar los paquetes RTCP, de los que extrae las siguientes métricas de calidad contenidas en los informes de receptor: *jitter* (jit) y fracción de paquetes perdidos desde el último informe (frpl). El AGR envía estos datos al SGR mediante *sockets* TCP. La periodicidad del envío la determina el protocolo RTCP. Se hace uso de Tshark [Tsk] para llevar a cabo el proceso de filtrado en el *host* cliente.

6.4.1.3. Servicio de Gestión de Recursos (SGR)

La función del SGR es recibir los informes desde el AGR, leer los datos incluidos en ellos y enviarlos al Buscador de Rutas. La comunicación entre los AGR y el SGR se efectúa mediante *sockets* TCP, mientras que las comunicaciones entre SGR y el controlador se realizan a través de la REST API del Buscador de Rutas.

El SGR se ha implementado también en un script en Python que gestiona los mensajes del protocolo PCMS e implementa métodos para enviar las métricas de calidad recibidas desde el AGR al Buscador de Rutas a través de llamadas REST. Para mantener correspondencia con la Figura 6.3, el AGR y el SGR se ejecutan como procesos paralelos a Video Tester.

6.4.2. Componentes del Controlador SDN

La arquitectura plantea incorporar dos módulos en el controlador de red, los que desempeñen funciones de monitorización y de cálculo de rutas óptimas

(MR y BR, respectivamente), como se muestra en la Figura 6.5. Estos módulos se sirven de los otros módulos internos del controlador; por ejemplo, para administrar las entradas de la tabla de flujo mediante el protocolo OpenFlow.

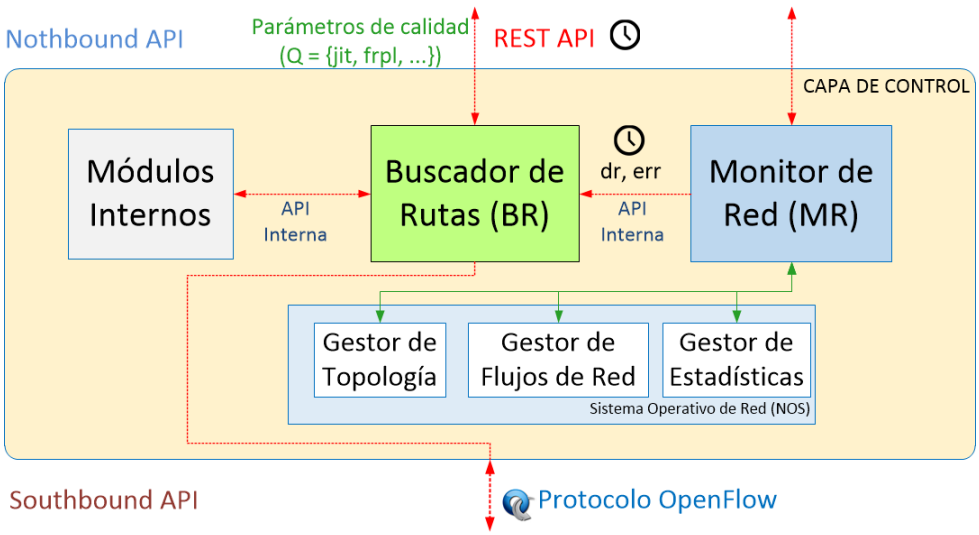

Figura 6.5: Componentes del controlador SDN.

Para la implementación de la arquitectura se ha utilizado Floodlight [Fdl] como sistema operativo de red.

6.4.2.1. Monitor de Red (MR)

Este módulo realiza el cálculo de parámetros como la tasa de datos y la tasa de error en períodos fijos de tiempo *T*. Un período consiste en un intervalo de tiempo comprendido entre los instantes *n-1* y *n*. Estos parámetros se miden en cada enlace de la topología y en cada par de puertos (*i, j*), conectados por un determinado enlace.

La tasa de datos del enlace (*dr*) se obtiene restando los bits enviados (*s*) en el momento de la monitorización menos los bits enviados al inicio del período en el puerto *i*. Este cálculo se muestra en la Ecuación 6.1.

$$dr_{i,j} = \frac{s_i^n - s_i^{n-1}}{T} \ [\text{bits/s}] \qquad (6.1)$$

La tasa de errores del enlace (*err*) se obtiene restando los bits recibidos (*r*) al cabo del período T en el puerto *j* menos los bits enviados (*s*) al cabo del período T en el puerto *i*. Este cálculo se muestra en la Ecuación 6.2.

$$err_{i,j} = \frac{(s_i^n - s_i^{n-1}) - (r_j^n - r_j^{n-1})}{T} \text{ [bits/s]} \qquad (6.2)$$

Se ha implementado este módulo para Floodlight. El período de monitorización T (expresado en milisegundos) está definido en la variable MR_STATS_REQUEST_INTERVAL_MS. El buscador de rutas solicitará al Monitor de Red (periódicamente) los valores de los últimos parámetros obtenidos mediante la API interna (Java) que presenta el módulo (Figura 6.5).

6.4.2.2. Buscador de Rutas (BR)

Este módulo es el encargado de encontrar la ruta óptima entre dos nodos (origen y destino) de la red. Utiliza el grafo G(N, A) que obtiene del Gestor de Topología para hallar la ruta más corta entre los nodos partícipes de la comunicación.

El Buscador de Rutas utiliza la información del Monitor de Red y los parámetros de calidad enviados por el SGR para calcular una métrica de ponderación que será utilizada como coste del enlace para el cálculo de la ruta más corta entre el origen y el destino de la comunicación. La ruta más corta se determinará aplicando el algoritmo de Dijkstra.

El módulo se ha implementado también sobre Floodlight y presenta una REST API para recibir los parámetros que le envíe el SGR (*jitter* y fracción de paquetes perdidos desde el último informe). Asimismo, usa la API interna de Java para solicitar al Monitor de Red la tasa de datos y la tasa de errores. Estos parámetros son solicitados en períodos de tiempo definidos en la variable BR_STATS_REQUEST_INTERVAL_MS y son almacenados para el cálculo de

las rutas. Se requiere, sin embargo, que este período sea mayor o igual al que utiliza el Monitor de Red para asegurar que se reciban siempre valores actualizados obtenidos en la última monitorización. Del mismo modo, el Buscador de Rutas almacena los últimos valores obtenidos desde el SGR (*jitter* y fracción de paquetes RTP perdidos desde el último informe), que luego utiliza en la función de coste.

El cálculo de la ruta se realiza aplicando el algoritmo de Dijkstra, asignando un coste a cada enlace del grafo. De esta manera, se extiende la métrica que usa el controlador Floodlight basada en evaluar únicamente el número de saltos entre origen y destino. El coste de cada enlace que une el nodo *i* con el nodo *j* (Ci,j) se obtiene aplicando la Ecuación 6.4:

$$C_{i,j} = \frac{dr_{i,j}}{bw_{i,j}} + \frac{err_{i,j}}{dr_{i,j}} + \alpha * (frpl/256) + \frac{jit(1-\alpha)}{MAX\_JITTER} \quad (6.4)$$

Donde:

- $C_{i,j}$: Coste del enlace i, j
- $dr_{i,j}$: Tasa de datos del enlace i, j
- $bw_{i,j}$: Ancho de banda del enlace i, j
- $err_{i,j}$: Tasa de errores del enlace i, j
- *frpl*: Fracción de paquetes RTP perdidos desde el último informe RTCP (está dividido entre 256 según la definición del protocolo RTCP.)
- *jit*: *Jitter* en la sesión RTP
- α: Factor de ponderación en la función ($0 \leq \alpha \leq 1$)
- *MAX_JITTER*: Valor máximo permisible de *jitter* en la transmisión.

Este módulo identifica los paquetes entrantes en el controlador utilizando la función *processPacketInMessage*. Los paquetes son enviados desde los conmutadores OpenFlow cada vez que encuentran un nuevo paquete cuyo encabezado no corresponde con ninguno de los criterios definidos en la tabla de flujo. En el controlador los paquetes de vídeo se identifican según los encabezados que se muestran en la Tabla 6.1:

Criterio	**Cliente**	**Servidor**
Protocolo de transporte	UDP	UDP
Puerto RTP	5532	X (Aleatorio, generado por Video Tester)
Puerto RTCP	5533	X+1
Dirección IP	10.0.0.2	10.0.0.1

Tabla 6.1: Paquetes de vídeo en el Buscador de Rutas.

Una vez recibido un paquete RTP según los criterios de la Tabla 6.1, el Buscador de Rutas analiza el grafo G(N, A) que representa la red, obtiene la ruta óptima y configura nuevas entradas en las tablas de flujo de los conmutadores OpenFlow que formen parte de la ruta encontrada. Las entradas en la tabla de flujo establecen como acción el reenvío del paquete hacia el próximo conmutador que forma parte de la ruta encontrada.

7. EXPERIMENTOS Y RESULTADOS

La evaluación del modelo consiste en demostrar la efectividad de la propuesta, enfocada en mejorar la calidad multimedia conseguida con la arquitectura propuesta con respecto a una transmisión SDN convencional.

7.1. Componentes Software

Para evaluar el modelo propuesto se han utilizado los componentes software descritos en la sección 6.3 y resumidos en la Tabla 7.1.

Componente	Herramienta	Lenguaje de programación
Cliente y servidor multimedia	Video Tester 1.0	Python 2.7.4
AGR y RMS	Implementación propia (script)	Python 2.7.4
NOS	Floodlight 0.90	Java
Monitor de Red, Buscador de Rutas	Implementación propia (módulos para Floodlight)	Java
Emulador del plano de datos	Impementación de scripts para Mininet 2.1.0	Python 2.7.4

Tabla 7.1: Componentes software para la evaluación del modelo.

7.2. Entorno de Ejecución

Los experimentos se han realizado en una máquina virtual Ubuntu Linux 13.04 (64 bits, 1500 MB de RAM, 1 CPU y 16 GB de disco duro). El plano de datos es emulado haciendo uso de Mininet [Mi1]. En el plano de control se usa Floodlight [Fdl] ejecutado sobre Eclipse [Ecl]. Simultáneamente, se ejecuta el software Video Tester [NOALS12] para implementar las funciones del cliente y servidor multimedia. En el cliente, y en paralelo, se ejecuta Tshark [Tsk] como

analizador de tráfico de red. A lo anterior se añade la ejecución de la máquina virtual de Python debido a que el AGR y el SGR se han programado como scripts en este lenguaje.

7.3. Experimentos

Se han efectuado pruebas para comprobar la efectividad del modelo. En primer lugar, se evalúa el impacto que tiene el funcionamiento del protocolo PCMS en función del tiempo de procesamiento requerido para el intercambio de mensajes previos al inicio de la transmisión.

En segundo lugar, se han obtenido los parámetros de evaluación de la calidad del vídeo al finalizar la transmisión. Para ello, se han utilizado los informes que ofrece Video Tester. La transmisión del vídeo se efectúa entre nodos de la red Mininet.

7.3.1. Topología de Prueba para el Protocolo PCMS

Se han implementado distintas topologías para evaluar el tiempo de ejecución requerido para procesar los mensajes del protocolo PCMS anteriores al inicio de la transmisión. Para este propósito las topologías de prueba se han generado utilizando las bibliotecas de Mininet en Python. Los distintos parámetros de configuración de las topologías se explican en la Tabla 7.2.

Parámetro	Significado
fanout	Es el número de hijos que tendrá cada nodo en la topología en árbol implementada. Los últimos nodos añadidos corresponden a los *hosts*; todos los anteriores son conmutadores OpenFlow. Se pasa como parámetro del constructor de una topología en árbol.
depth	Es el nivel de profundidad del árbol. Se pasa como parámetro del constructor de una topología en árbol.
TreeNet	Es una clase de Mininet para generar una topología en árbol, según los parámetros *fanout* y *depth* establecidos en el constructor.
CustomTree	Es una clase personalizada que crea una topología en árbol introduciendo, además del *fanout* y *depth*, parámetros de ancho de banda, retardo y tasa de errores en los enlaces de la topología. Esta clase se ha implementado extendiendo la clase Topo del módulo *mininet.topo* en Python.
bw	Ancho de banda de un enlace creado, expresado en Mbps.
delay	Retardo de un enlace creado, expresado en ms.
loss	Tasa de errores del enlace, expresado en porcentaje.
default options	Opciones por defecto que usa Mininet para generar los enlaces de las topologías.

Tabla 7.2: Parámetros de configuración de topologías en Mininet.

La Figura 7.1 muestra un ejemplo de topología creada con la clase *TreeNet* (*fanout*=2, *depth*=3) de Mininet.

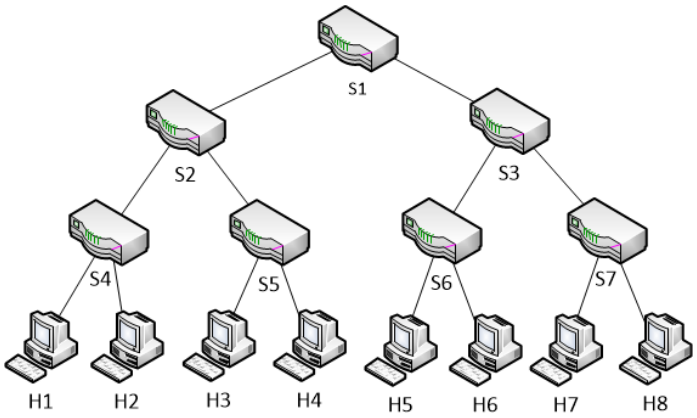

Figura 7.1: Topología creada con la clase TreeNet.

Se ha experimentado con distintas topologías de árbol (binario o ternario) implementadas en Mininet y detalladas en la Tabla 7.3. En cada una se han tomado los tiempos de procesamiento del SGR y del AGR, requeridos para intercambiar todos los mensajes previos al inicio de la transmisión del contenido multimedia, detallados en la sección 6.3.1.

Caso de prueba Nº	Topología	SGR (ms)	AGR (ms)
1	Conexión directa de red	3,2876	3,9998
2	Nat (Mininet TreeNet) [fanout=2, depth=1, default options] (1 conmutador)	3,0516	3,6409
3	Nat (Mininet TreeNet) [fanout=2, depth=3, default options] (7 conmutadores)	3,3313	4,0753
4	Nat (Mininet TreeNet) [fanout=2, depth=5, default options] (31 conmutadores)	3,2146	3,9932
5	Nat (Mininet TreeNet) [fanout=2, depth=7, default options] (127 conmutadores)	3,9415	5,2681
6	Nat (Mininet TreeNet) [fanout=3, depth=5, default options] (121 conmutadores)	3,6874	3,9663
7	Nat (CustomTree) [fanout=3, depth=5, default options] (121 conmutadores)	3,9815	4,3584
8	Nat (CustomTree) [fanout=3, depth=5, bw: 10-30 Mbps] (121 conmutadores)	5,2159	5,5463
9	Nat (CustomTree) [fanout=3, depth=5, bw: 10-30 Mbps, delay=1ms] (121 conmutadores)	47,6448	48,1503
10	Nat (CustomTree) [fanout=3, depth=5, bw: 10-30 Mbps, delay=0.5ms, loss=1] (121 conmutadores)	362,328	361,032

Tabla 7.3: Topologías y tiempos de procesamiento en el RMS y AGR.

En todos los casos, con excepción del primero, se han construido topologías con un número variable de conmutadores y se ha situado al servidor SGR fuera de la red Mininet (a través de un esquema NAT). En la Figura 7.2 se muestra una representación gráfica de la topología utilizada en el caso de prueba número 8.

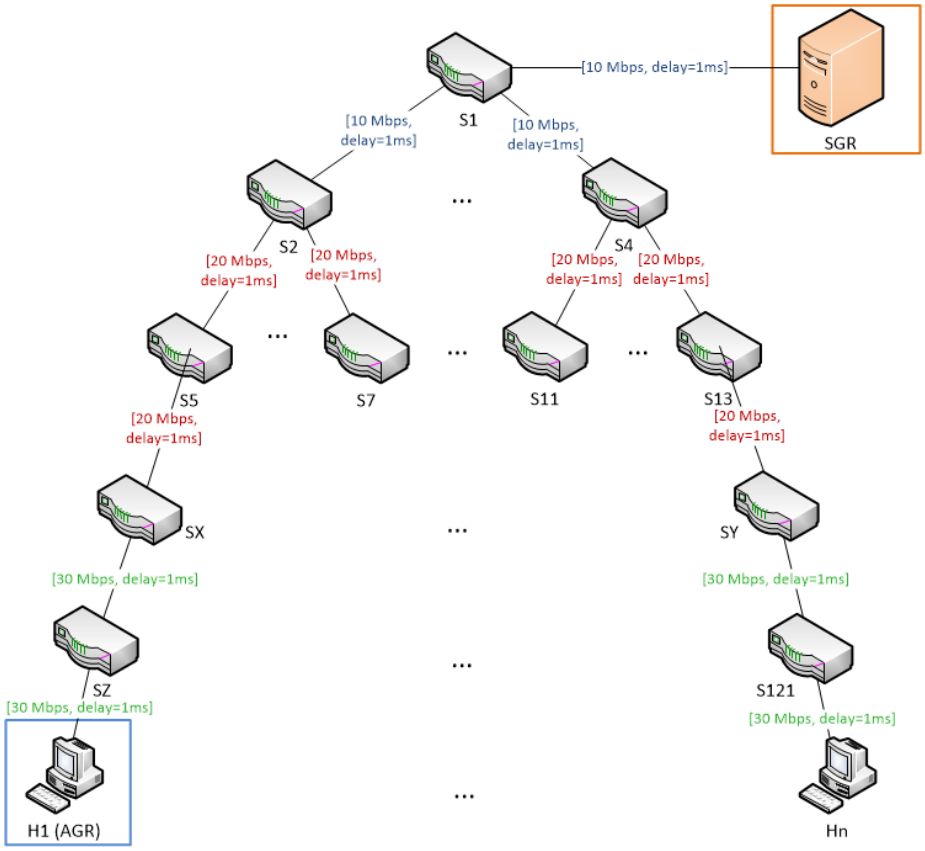

Figura 7.2: Topología creada en el caso de prueba nº 8.

Se ha evaluado por separado el tiempo de procesamiento en el SGR y en el AGR. Los valores expresados en milisegundos reflejan el valor promedio obtenido de un bloque de 100 conexiones.

Con los resultados obtenidos se puede apreciar que en circunstancias de operación normales no se aprecia un impacto significativo en el rendimiento de la red. No se han superado los 5 milisegundos de tiempo de procesamiento en los casos de operación más típicos (casos 1 al 8), incluso con un número elevado de conmutadores (121). Sólo en condiciones de operación complejos (casos 9 y 10), cuando se introducen valores de retardo y tasa de errores en los distintos niveles de la jerarquía de árbol, los valores de procesamiento se ubican entre los 47 y los 361 ms.

7.3.2. Topología de Prueba para la Transmisión de Vídeo

Para la transmisión del vídeo se ha implementado una topología de cinco conmutadores OpenFlow, que se muestra en la Figura 7.3. Se han dispuesto los enlaces de tal forma que el algoritmo del buscador de rutas los selecciona considerando el coste de cada enlace, como se vio en el apartado 6.4.2.2. Se ha optado por implementar esta topología por cuestiones de rendimiento del entorno de ejecución, debido a que la máquina virtual viene ejecutando en simultáneo distintas herramientas software necesarias para las pruebas (detalladas en la sección 7.2) que repercuten en el rendimiento total del sistema, limitando así la implementación de topologías más complejas.

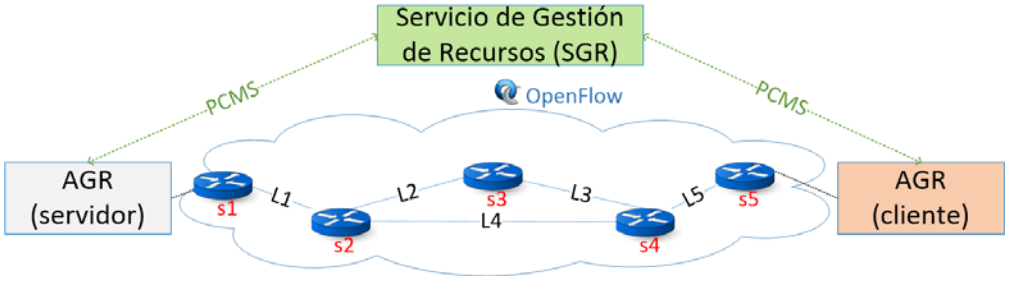

Figura 7.3: Topología de prueba para la transmisión de vídeo.

En los enlaces dispuestos en la topología se han variado los parámetros de operación (tasa de transferencia, retardo y tasa de errores), como se muestra en la Tabla 7.4.

Enlace	Tasa de transferencia (Mbps)	Tasa de errores (%)	Retardo (ms)
L1,L2,L3,L5	20	0	0
L4	20	2	2

Tabla 7.4: Parámetros de cada enlace en la topología de prueba.

Se han utilizado los componentes software cuya implementación ha sido descrita en el capítulo anterior para evaluar la calidad en la transmisión de un

vídeo de prueba [Vid], cuyas características se muestran en la Tabla 7.5.

Característica	Valor
Códec	H264 MPEG-4
Resolución	352 x 290
Formato decodificado	*Planar* 4:2:0 YUV
Tasa de fotogramas por segundo	25

Tabla 7.5: Característica del vídeo empleado en las pruebas.

El vídeo referenciado se transmite por la red SDN mediante el protocolo RTP. Para valorar la calidad del vídeo recibido la herramienta Video Tester realiza la descompresión del archivo (a formato .yuv), del cual obtiene los valores necesarios para obtener los valores de *Peak Signal-to-Noise Ratio* (PSNR) y de *Mean Opinion Score* (MOS) [GKKW04], descritos en la sección 5.3.

La función de coste aplicada se ha establecido teniendo en cuenta la Ecuación 6.4, estableciéndose el valor de α en 0.7 y el *jitter* máximo permisible (*MAX_JITTER*) en 1000.

7.4. Resultados

En primer lugar se ha medido el valor del PSNR. Para ello se han usado los informes de Video Tester que se emiten al finalizar la transmisión. En la Figura 7.4 se muestra el valor del PSNR obtenido en un escenario de operación que no contempla la arquitectura propuesta, es decir, simulando la transmisión en la red Mininet con el controlador de red externo (Floodlight) y ejecutando Video Tester en ambos extremos de la comunicación (cliente y servidor). En la Figura 7.5 se muestra la gráfica del PSNR obtenido después de implementar la arquitectura planteada en este trabajo, contemplando la presencia de todos los componentes explicados en el capítulo anterior.

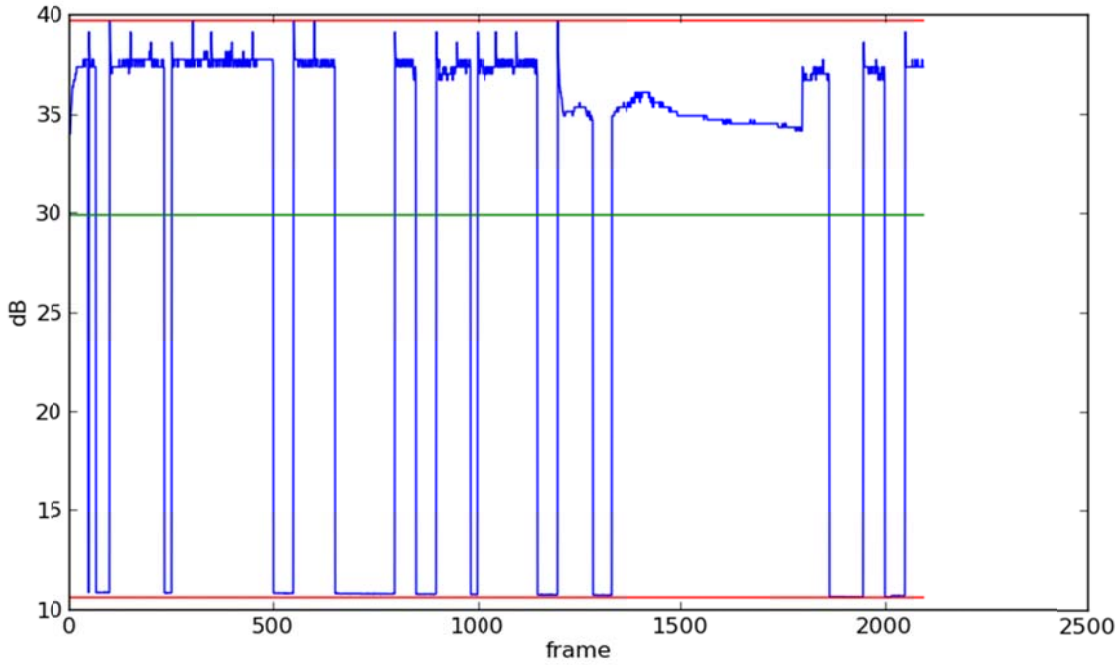

Figura 7.4: Medida del PSNR sin la arquitectura propuesta.

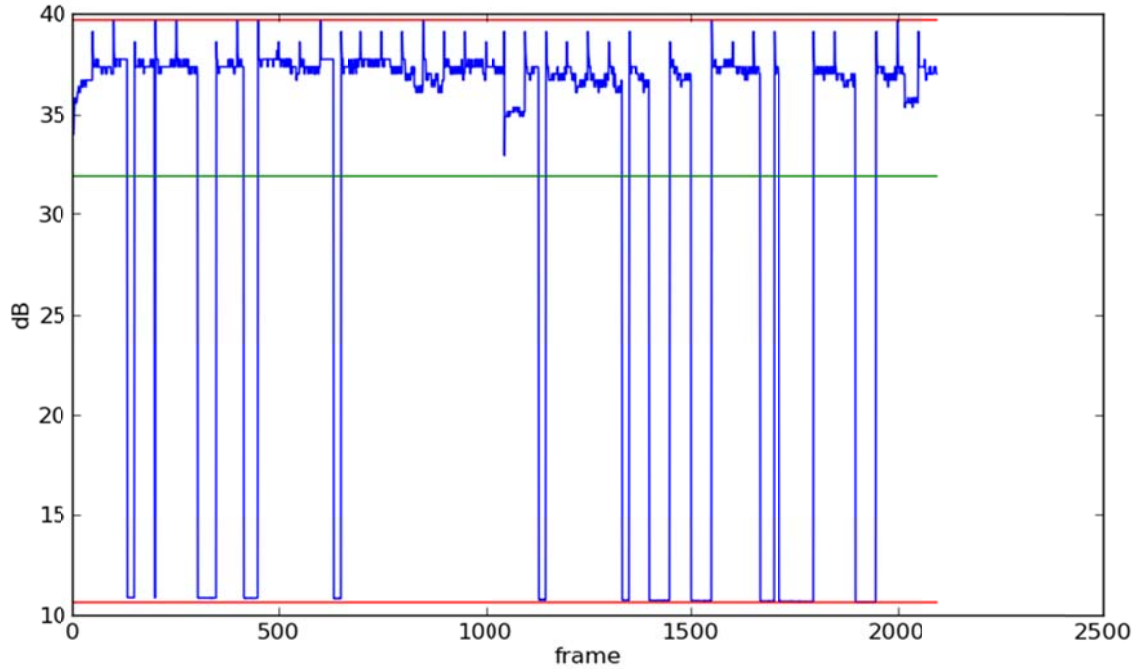

Figura 7.5: Medida del PSNR después de implementar la arquitectura.

En ambas figuras, el eje de abscisas representa los fotogramas del vídeo y el eje de ordenadas el valor de PSNR medido en decibelios (dB). Se puede ver en la segunda gráfica que existe un incremento tanto en el número de decibelios máximos alcanzados (línea roja) como en el valor promedio resultante (línea verde).

Como en el caso del PSNR, se presentan las evaluaciones obtenidas del MOS con Video Tester en las Figuras 7.6 y 7.7, que corresponden a su medición en dos escenarios, sin la arquitectura y con la arquitectura implementada, respectivamente.

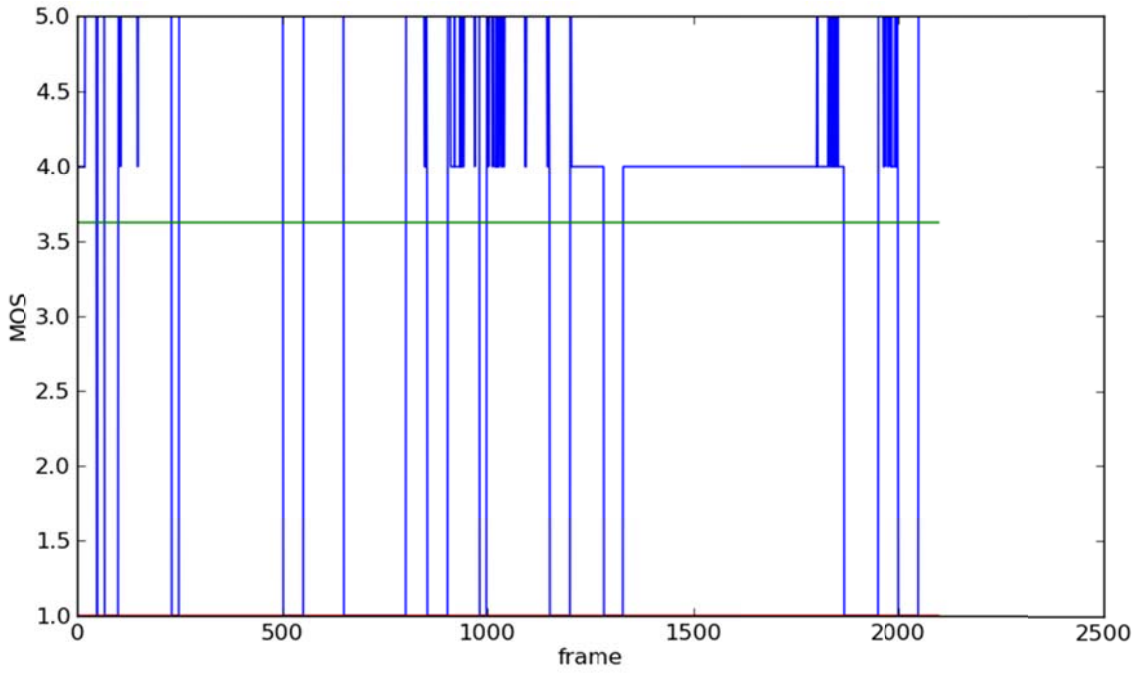

Figura 7.6: Medida del MOS sin la arquitectura propuesta.

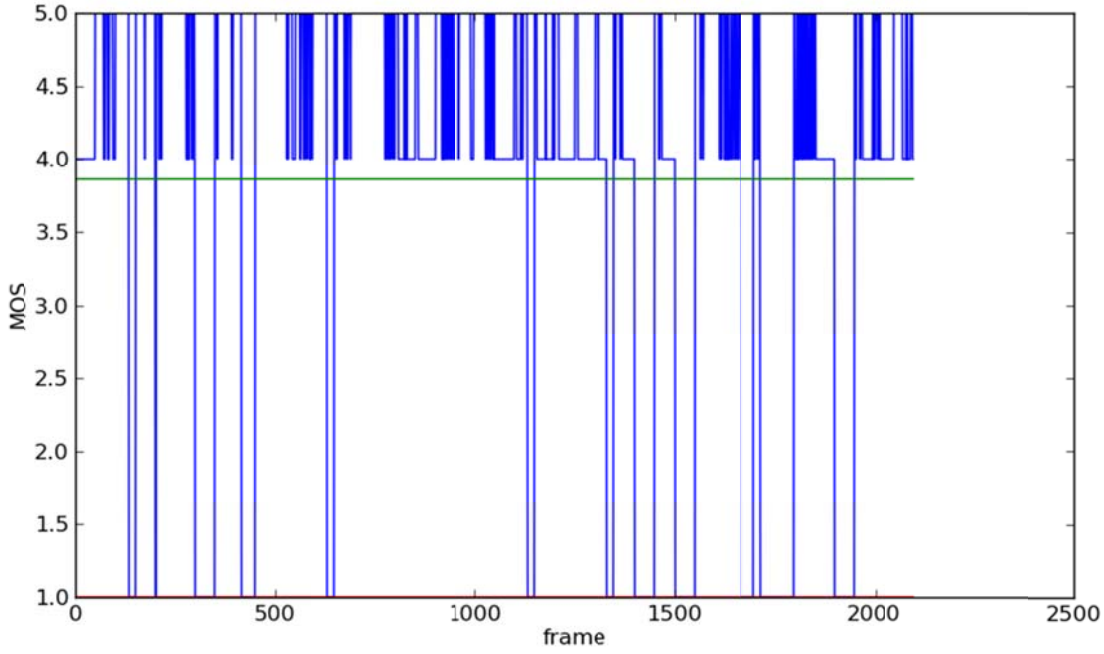

Figura 7.7: Medida del MOS luego de implementar la arquitectura.

Como pudo verse en la sección 5.3, a partir del valor PSNR se puede obtener una medida del MOS. Video Tester se encarga de hacer esa conversión y consolidar esos resultados.

El MOS promedio (línea verde) se ve incrementado en la Figura 7.6 después de implementarse la arquitectura, lo que se traduce en un indicador de mejora en la percepción de calidad del usuario.

Según se ha podido observar en estas comparativas, es posible incrementar la calidad de los recursos multimedia transmitidos si se implementa la arquitectura propuesta. Se ha dispuesto de una topología propensa a errores y a factores de retardo inducidos que permiten evaluar el modelo con mayor ajuste a condiciones del mundo real, en las que el comportamiento de la red es poco controlable.

8. CONCLUSIONES Y TRABAJO FUTURO

8.1. Conclusiones

Como resultado de la presente investigación se ha comprobado que la separación de los planos de infraestructura, control y aplicación de la arquitectura SDN hace posible incorporar información perteneciente a dichos planos como entradas para establecer el criterio de encaminamiento de la red, aplicado a un conjunto específico de paquetes multimedia. Del plano de aplicación se ha obtenido información de calidad de servicio multimedia proporcionada por el protocolo RTCP. Del plano de infraestructura se extrae la información necesaria para calcular parámetros de estado de la red en el plano de control. Con ese conjunto de datos el controlador SDN realiza el cálculo de una métrica combinada que considera la información proporcionada por cada plano de la arquitectura.

El análisis de calidad del contenido multimedia, posterior a la transmisión de datos y con la ayuda de una herramienta especializada, ha podido mostrar una mejora en la calidad experiencia del usuario. Para ello se ha considerado una métrica objetiva como el PSNR, y de ella se ha derivado el valor de una métrica subjetiva como el MOS. En base a estas dos métricas se ha podido evaluar la mejora de la calidad de experiencia del usuario, comprobándose que las decisiones de encaminamiento en el plano de datos afectan a la calidad del contenido multimedia procesado en capas superiores.

Para llevar a cabo las tareas de monitorización se ha implementado un módulo de software, llamado Monitor de Red (MR), que obtiene métricas del plano de datos. Esas métricas son aprovechadas por el módulo Buscador de Rutas (BR) que las procesa conjuntamente con los parámetros de calidad obtenidos desde los Agentes de Gestión de Recursos (AGR) para aplicar una

función de coste personalizada, considerando factores de ajuste para la ponderación de la tasa de errores a nivel de aplicación y el *jitter* máximo permisible para una transmisión multimedia en particular.

Frente a la necesidad de comunicar componentes software a través de una red se ha implementado un protocolo de comunicación, denominado PCMS. Ello demuestra el nivel la flexibilidad que otorga SDN para que los propios administradores de una red personalicen el comportamiento de la red y sus componentes, y que esa lógica personalizada se traduzca en reglas de encaminamiento aplicables en el plano de datos. En las redes TCP/IP convencionales los procesos de propuesta o estandarización de nuevos protocolos de red puede tomar mucho tiempo, y su adopción por la industria aún más.

La arquitectura propuesta en este trabajo no se limita a un conjunto específico de protocolos ni métricas de análisis en particular, sino que deja abiertas las posibilidades de integrar distintos esquemas de transmisión multimedia, siempre que se incorporen, como mínimo, los componentes software definidos en la propuesta y se asegure la existencia de interfaces de comunicación entre ellos. Se espera que en futuros trabajos se obtengan más resultados de la aplicación del modelo a entornos multimedia distintos, mediante el uso de nuevas métricas, protocolos y criterios de encaminamiento en el controlador SDN.

Una solución como la que se ha presentado en esta investigación puede ser aprovechada por un administrador de red, para que pueda contar con un entorno más controlado que le permita cumplir los niveles de calidad de servicio acordados con los usuarios.

El diseño de la arquitectura presentada responde a un contexto específico como el de transmitir contenido multimedia. Teniendo en cuenta aquello, se han propuesto los componentes software necesarios y el protocolo de

comunicación PCMS. Sin embargo, y por las características de las redes SDN (separación de funciones y diseño modular del plano de control) se considera factible construir arquitecturas similares útiles para otros contextos.

8.2. Trabajo Futuro

Existen numerosas aplicaciones SDN para dar solución a las limitaciones que muestran las redes de comunicación convencionales. A lo largo de esta investigación se ha explorado el escenario de transmisión multimedia, pero quedan abiertos muchos otros escenarios, desde los más generales hasta otros más específicos.

Para enriquecer la interoperabilidad de los componentes software definidos en la arquitectura se puede modificar la estructura de los mensajes del protocolo PCMS para gestionar múltiples comunicaciones multimedia simultáneas entre diferentes orígenes y destinos. En esta investigación se ha asumido que el AGR y el SGR procesan mensajes procedentes de una única fuente, pero se pretende que procesen más de una a la vez. Incluir las direcciones IP de cada participante en el formato del mensaje o el identificador de fuente que proporciona el protocolo RTP puede servir para ese fin. El soporte a comunicaciones simultáneas entre los componentes requiere también que el Buscador de Rutas (BR) almacene los parámetros de calidad procedentes de los SGR en tablas separadas por cada par de nodos origen y destino.

Es deseable también validar el modelo comparativamente con más protocolos multimedia y con casos de prueba más complejos que puedan ayudar a demostrar la efectividad del modelo en tales escenarios.

También es una labor pendiente extender la topología de pruebas, superando para ello las limitaciones del entorno de ejecución que se comentaron al inicio de la fase de experimentación.

No se han asumido mecanismos de tolerancia a fallos en los componentes software implementados en la arquitectura. De este modo, se han obtenido los resultados en condiciones de operación normal en la red, sin considerar, por ejemplo, la caída del controlador SDN o algún tipo de error en los AGR o en el SGR.

Los módulos del controlador de red son también susceptibles de mejoras. En el Monitor de Red se puede definir un esquema más óptimo que evite recopilar datos estadísticos de las tablas de flujo asociadas a cada par de conmutadores OpenFlow pertenecientes a un enlace, lo que puede requerir mucho tiempo en redes grandes. En su lugar, se puede optar por mecanismos capaces de abreviar este proceso. En el Buscador de Rutas (BR) se puede optimizar el período de consulta al Monitor de Red, y mejorar el criterio de encaminamiento y asignación de entradas de flujo en los conmutadores OpenFlow.

Finalmente, queda por investigar si se puede extender la arquitectura SDN a escenarios más diversos o generalizar el rol de los componentes software, propuestos para dar respuesta a otras necesidades de mejora en los procesos de comunicación sobre una red SDN.

CONCLUSIONS AND FUTURE WORK

Conclusions

As a result of this work it has been demonstrated that the separation of the control, application and data planes in the SDN architecture makes it possible to incorporate information belonging to those planes as inputs to set a routing criteria of the network, applied to a specific set of multimedia packets. From the application plane it has been obtained information about multimedia quality of service, given by the RTCP protocol. From the infrastructure plane it is extracted the required information to compute network state parameters in the control plane. With this data set, the SDN controller computes a combined metric that considers the information provided by each plane of the architecture.

Quality analysis of multimedia content, performed after the data transmission and assisted by a specialized tool, has shown an improvement in the user quality of experience. For that, it has been taken an objective metric as PSNR, and from it the value of a subjective metric like the MOS value has been derived. Based on these two metrics, the improvement in the user quality of experience was evaluated. As a result, it has been proved that routing decisions taken in the data plane affect the quality of multimedia content processed in the upper layers.

To perform the monitoring tasks it was implemented a software module, called Network Monitor (MR), which gathers metrics from the data plane. These metrics are used by the Route Finder module (BR) that process them along with the quality parameters obtained from the Resource Management Agents (AGR) to apply a personalized cost function, considering adjustment factors to weight the error rate in the application level as well as the maximum permitted jitter for a particular multimedia transmission.

To cope with the need to communicate software components over a network it has been implemented a communication protocol, called PCMS. It demonstrates the level of flexibility that SDN gives to network administrators, in order to allow them customizing the behavior of the network and its components, and translates this personalized logic into routing directives applicable in the data plane. In conventional TCP/IP networks the process of propose or standardize new network protocols could take a lot of time, and even more its adoption by the industry.

The proposed architecture in this work it is not limited neither to a specific set of protocols nor particular analysis metrics. Instead of that, leaves open possibilities to integrate diverse communication and multimedia transmission schemes, provided that, at least, the software components defined in this proposal are integrated and the existence of communication interfaces between them is assured. It is expected that future works will obtain more results due to the application of the model to several multimedia environments, through the use of new metrics, protocols and routing criteria in the SDN controller.

A solution like the one presented in this work could benefit a network administrator, in order to have a more controlled environment that allows meeting the service quality levels agreed with the customers.

The design of the proposed architecture responds to a specific context as is the transmission of multimedia content. Bearing this in mind, the required software components and the PCMS communication protocol have been proposed. However, and due to the characteristics of SDN networks (separation of functions and modular design of the control plane) it is considered to be feasible building similar architectures useful to other contexts.

Future Work

Several SDN applications exist to overcome the limitations shown by the conventional communication networks. Along this work, the scenario of multimedia transmission has been explored, but many other research scenarios remain opened, from the more general ones to more specific others.

To enrich the interoperability of the software components defined in the architecture it is possible to modify the structure of the PCMS protocol messages in order to manage multiple multimedia communications simultaneously between different sources and destinations. In this work, it has been assumed that the AGR and SGR compute messages coming from a unique source, but the processing of more than one at the same time is expected. The incorporation of IP addresses from each participant in the message format or source identifier provided by RTP protocol can be useful to this purpose. The support for multiple multimedia communications requires also that the Route Finder (BR) stores a quality parameters coming from the SGR in separated tables for each pair of source and destination nodes.

It is desirable also to validate the model comparatively with more multimedia protocols and more complex test cases that can help to demonstrate the effectiveness of the model in such scenarios.

It is also a pending task to extend the test topology, overcoming for that purpose the limitations of the execution environment commented at the beginning of the experimentation phase.

There have not been assumed fault tolerance mechanisms in the software components implemented in the architecture. This way, the results have been obtained in normal operational conditions in the network, not considering, for example, a SDN controller crash or some kind of error either in the AGR or in the SGR.

The controller modules are also susceptible to improvements. The Network Monitor (MR) can define a more optimal scheme that avoids gathering statistics from every flow table associated to a pair of OpenFlow switches belonging to a link, which can be time consuming in big networks. Instead of it, it is possible to choose for mechanisms able to abbreviate that process. In the Route Finder (BR) it is possible to optimize the time interval of queries to the Network Monitor, and improve the routing criteria and the assignment of flow table entries in the OpenFlow switches.

Finally, it is pending to explore if it is possible to extend the SDN architecture to a more diverse scenarios or to generalize the role of the proposed software components in order to respond to other improvement needs in the communication processes over an SDN network.

BIBLIOGRAFÍA

[Azo13] S. Azodolmolky. *Software Defined Networking with OpenFlow*. Packt Publishing Ltd, October 2013.

[Bsn] Big Switch Networks. http://www.bigswitch.com.

[CCF+05] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Van der Merwe. Design and Implementation of a Routing Control Platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 15-28, Boston, Massachusetts, May 2005.

[CFP+07] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 1-12, Kyoto, Japan, August 2007.

[Cis15] Cisco. The Zettabyte Era-Trends and Analysis. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html.

[DHM+13] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella. Towards an Elastic Distributed SDN Controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 7-12, Hong Kong, China, August 2013.

[Ecl] The Eclipse Foundation Open Source Community. https://www.eclipse.org.

[Fdl] Project Floodlight - Floodlight SDN Controller. http://www.projectfloodlight.org/floodlight.

[FRZ13] N. Feamster, J. Rexford, and E. Zegura. The Road to SDN. *Queue*, 11(12):20, December 2013.

[GCRVACCB14] A. García Centeno, C. M. Rodríguez Vergel, C. Anías Calderón, and F. C. Casmartiño Bondarenko. Controladores SDN: Elementos para su Selección y Evaluación. *Revista Telemática*, 13(3):10-20, December 2014.

[GKKW04] J. Gross, J. Klaue, H. Karl, and A. Wolisz. Cross-Layer Optimization of OFDM Transmission Systems for MPEG-4 Video Streaming. *Computer Communications*, 27(11):1044-1055, July 2004.

[GKP+08] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105-110, July 2008.

[Gst] GStreamer: Open Source Multimedia Framework. http://gstreamer.freedesktop.org.

[GVS+14] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett. SDX: A Software Defined Internet Exchange. In *Proceedings of the ACM conference on SIGCOMM*, pages 551-562, Chicago, Illinois, August 2014.

[HSM+10] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving Energy in Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, pages 1-16, San Jose, California, April 2010.

[KF13] H. Kim and N. Feamster. Improving Network Management with Software Defined Networking. *IEEE Communications Magazine*, 51(2):114-119, February 2013.

[Koi00] T. Koistinen. Protocol Overview: RTP and RTCP. Technical report, Nokia Telecommunications, 2000.

[KRW03] J. Klaue, B. Rathke, and A. Wolisz. EvalVid - A Framework for Video Transmission and Quality Evaluation. In *Computer Performance Evaluation. Modelling Techniques and Tools*, volume 2794 of *Lecture Notes in Computer Science*, pages 255-272. Springer, 2003.

[KSKD+12] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely. Towards QoE-Driven Multimedia Service Negotiation and Path Optimization with Software Defined Networking. In *Proceedings of the 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1-5, Split, Croatia, September 2012.

[KZMB14] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou. Feature-Based Comparison and Selection of Software Defined Networking (SDN) Controllers. In *Proceedings of the IEEE World Congress on Computer Applications and Information Systems (WCCAIS)*, pages 1-7, Hammamet, Tunisia, January 2014.

[LCMP+13] P. Le Callet, S. Möller, A. Perkis, et al. Qualinet White Paper on Definitions of Quality of Experience. White paper, European Network on Quality of Experience in Multimedia Systems and Services, March 2013.

[LHM10] B. Lantz, B. Heller, and N. McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1-6, Monterey, California, October 2010.

[LWH+12] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. Logically centralized?: State Distribution Trade-Offs in Software Defined Networks. In *Proceedings of The First ACM SIGCOMM Workshop on Hot Topics in Software Defined Networks*, pages 1-6, Helsinki, Finland, August 2012.

[MAB+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69-74, April 2008.

[Mi1] Mininet: An Instant Virtual Network on your Laptop (or other PC). http://mininet.org.

[MVTG14] J. Medved, R. Varga, A. Tkacik, and K. Gray. Opendaylight: Towards a Model-Driven SDN Controller Architecture. In *Proceedings of the IEEE 15th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1-6, Sydney, Australia, June 2014.

[Nc1] NEC Corporation. http://www.nec.com.

[NG13] T. D. Nadeau and K. Gray. *SDN: Software Defined Networks*. O'Reilly Media, Inc., September 2013.

[NMN+14] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, T. Turletti, et al. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617-1634, February 2014.

[NOALS12] J. Navarro-Ortiz, P. Ameigeiras, and J. M. Lopez-Soler. Video Tester-A Multiple-Metric Framework for Video Quality Assessment over IP Networks. In *Proceedings of the IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, pages 1-5, Seoul, Korea, June 2012.

[Nx1] NOX SDN Controller. http://www.noxrepo.org.

[Odl] Open Daylight Platform. http://www.opendaylight.org.

[Op1] Open Networking Foundation. https://www.opennetworking.org.

[Ope09] Open Networking Foundation. OpenFlow Switch Specification version 1.0.0. Technical specification, ONF, December 2009.

[Ope12a] Open Networking Foundation. OpenFlow Switch Specification version 1.3.0. Technical specification, ONF, June 2012.

[Ope12b] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, ONF, April 2012.

[Ost] OpenStack: Open Source Cloud Computing Software. http://www.openstack.org.

[PF14] J. A. Puente Fernández. Algoritmo de Calidad de Experiencia para Transmisiones de Vídeo en Redes Definidas por Software. Trabajo de fin de Máster, Universidad Complutense de Madrid, 2014.

[Pri15a] Princeton University. Software Defined Networking Course: Control and Data Plane Separation, 2015.

[Pri15b] Princeton University. Software Defined Networking Course: History and Evolution of SDN, 2015.

[Rib07] A. F. Ribadeneira. An Analysis of the MOS Under Conditions of Delay, Jitter and Packet Loss and an Analysis of the Impact of Introducing Piggybacking and Reed Solomon FEC for VoIP. Master thesis, Georgia State University, 2007.

[Ry1] Ryu SDN Framework. http://osrg.github.io/ryu.

[SCFJ03] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, Internet Engineering Task Force, July 2003.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). RFC 2326, Internet Engineering Task Force, April 1998.

[SSHC+13] S. Sezer, S. Scott-Hayward, P. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen,M. Miller, and N. Rao. Are we ready for SDN? Implementation Challenges for Software-Defined Networks. *IEEE Communications Magazine*, 51(7):36-43, July 2013.

[SYF+15] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker. Hone: Joint Host-Network Traffic Management in Software-Defined Networks. *Journal of Network and Systems Management*, 23(2):374-399, April 2015.

[TGG10] A. Tootoonchian, M. Ghobadi, and Y. Ganjali. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *Passive and Active Measurement*, volume 6032, pages 201-210. Springer, January 2010.

[Trea] Trema Applications. https://github.com/trema/apps.

[Treb] Trema: Full-Stack OpenFlow Framework in Ruby and C. http://trema.github.io/trema.

[Tsk] Tshark: Terminal-Based Wireshark. https://www.wireshark.org/docs/man-pages/tshark.html.

[VADK14] N. L. M. Van Adrichem, C. Doerr, and F. A. Kuipers. OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks. In *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*, pages 1-8, Krakow, Poland, May 2014.

[VCBPBLGV14] A. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona Lopez, and L. J. García Villalba. SDN: Evolution and Opportunities in the Development IoT Applications. *International Journal of Distributed Sensor Networks*, 2014, May 2014.

[Vid] Video Utilizado en las Pruebas de Transmisión. http://www2.tkn.tuberlin.de/research/evalvid/cif/bridge-far_cif.264.

[Zur04] R. Zurawski. *The Industrial Information Technology Handbook*. CRC Press, November 2004.